UNISYS

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D FINAL REPORT
Unisys Defense Systems
Contract Number: N00014-85-C-0012

Volume III -- PAPERS

ARPA ORDER NUMBER: 5262
PROGRAM CODE NO. NR 049-602 dated 10 August 1984 (433)
CONTRACTOR: Unisys/Defense Systems
CONTRACT AMOUNT: 1,704,901
CONTRACT NO: N00014-85-C-0012
EFFECTIVE DATE OF CONTRACT: 4/29/85
EXPIRATION DATE OF CONTRACT: 9/30/89
PRINCIPAL INVESTIGATOR: Dr. Lynette Hirschman PHONE NO. (215) 648-7554

SHORT TITLE OF WORK: DARPA Natural Language Understanding Program

REPORTING PERIOD: 4/29/85-9/30/89

UNISYS

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D FINAL REPORT
Unisys Defense Systems
Contract Number: N00014-85-C-0012

Volume III -- PAPERS

ARPA ORDER NUMBER: 5262
PROGRAM CODE NO. NR 049-602 dated 10 August 1984 (433)
CONTRACTOR: Unisys/Defense Systems
CONTRACT AMOUNT: 1,704,901
CONTRACT NO: N00014-85-C-0012
EFFECTIVE DATE OF CONTRACT: 4/29/85
EXPIRATION DATE OF CONTRACT: 9/30/89
PRINCIPAL INVESTIGATOR: Dr. Lynette Hirschman PHONE NO. (215) 648-7554

SHORT TITLE OF WORK: DARPA Natural Language Understanding Program

REPORTING PERIOD: 4/29/85-9/30/89

TABLE OF CONTENTS - VOL. III PAPERS

Ball, C. "Analysing explicitly-structured discourse in a limited domain: trouble and failure reports". In *Proceedings of the DARPA Workshop on Speech and Language*, March 1989.

ANALYZING EXPLICITLY-STRUCTURED DISCOURSE IN A LIMITED DOMAIN: TROUBLE AND FAILURE REPORTS*

Catherine N. Ball
Unisys Corporation
Paoli Research Center
P.O. Box 517, Paoli PA 19301

ABSTRACT

Recent theories of focusing and reference rely crucially on discourse structure to constrain the availability of discourse entities for reference, but deriving the structure of an arbitrary discourse has proved to be a significant problem. A useful level of problem reduction may be achieved by analyzing discourse in which the structure is explicit, rather than implicit. In this paper we consider a genre of explicitly-structured discourse: the Trouble and Failure Report (TFR), whose structure is both explicit and constant across discourses. We present the results of an analysis of a corpus of 331 TFRs, with particular attention to discourse segmentation and focusing. We then describe how the Trouble and Failure Report was automated in a prototype data collection and information retrieval application, using the PUNDIT natural-language processing system.

INTRODUCTION

Recent theories of focusing and reference rely crucially on discourse structure to constrain the availability of discourse entities for reference, but deriving the structure of an arbitrary discourse has proved to be a significant problem ([Webber 88]). While progress has been made in identifying the means by which speakers and writers mark structure ([Grosz 86], [Hirschberg 87], [Schiffrin 87], [Webber 88]), much work remains to be done in this area.

As is well known, initial progress in computational approaches to syntax and semantics was facilitated by reducing the problem space to discourses in technical sublanguages, in simplified registers, in restricted domains[1]. For Computational Pragmatics, the analysis of explicitly-structured discourse can provide a similar level of problem reduction. By removing the theoretical obstacle of deriving discourse structure, we can more readily evaluate the effect of this structure on focusing and reference.

In this paper we consider a genre of explicitly-structured discourse, namely the 'form', in which each labelled box and the response within it constitute a discourse segment. From the perspective of discourse understanding, the study of forms-discourse offers considerable advantages: the structure of the form is pre-defined and constant across discourses, and it is possible to study patterns of reference in narrative responses without excessive reliance on intuition. The particular form which we consider here is the Trouble and Failure Report (TFR). We first discuss the results of an analysis of 331 TFRs, and then describe the implementation of a TFR analysis module using the PUNDIT natural-language processing system.

THE TROUBLE AND FAILURE REPORT

TFRs are used to report problems with hardware, software, or documentation on equipment on board Trident and Poseidon submarines. These reports originate on board the submarine, and those concerning the Navigational Subsystem (which is managed by the Unisys Logistics Group) are routed to Unisys for analysis and response.

*This work has been supported by DARPA contract N00014-85-C-0012, administered by the Office of Naval Research.
[1]See for example the papers in [Grishman 86b].

The TFR contains a formatted section and up to 99 lines of free text. The formatted section includes coded information identifying the message originator, date, equipment, and failed part. The free text is divided into 5 sections, labelled A-E, each of which documents a specific aspect of the problem being reported. A sample hardware TFR is given below[2]:

<Formatted lines...>

A. WHILE PERFORMING SDC 955Z (GENERATION OF LASER BEAMS) TRANSPORTER UPPER TRANSLOCK WENT OFF LINE. B. UPPER TRANSLOCK INTERPORT SWITCH WENT BAD, UNABLE TO RE-ENERGIZE ETHER REGULATOR IN UPPER TRANSLOCK WHEN INTERPORT SWITCH DEPRESSED. C. DETERIORATION DUE TO AGE AND WEAR. D. REPLACED INTERPORT SWITCH WITH A NEW ONE FROM SUPPLY. E. NONE.

TFRs are stored in a historical database. Although the formatted data can be mapped to specific fields of database records, which can then be accessed by a query language, the free-text portions are stored as undigested blocks of text. Currently, keyword search is the only method by which the text can be accessed. Problems with keyword search as a method of information retrieval are well-known[3], and this is an area in which NLP techniques can be applied, with potential benefits of increasing the efficiency and accuracy of information retrieval.

As part of an internally and DARPA-funded R&D project, we applied PUNDIT ([Grishman 86a], [Dahl 87], [Dahl 86]) to the analysis of TFRs. Previous applications of PUNDIT to the analysis of the **Remarks** field of Navy messages had required only a superficial level of discourse processing above the paragraph. But the richer discourse structure of TFRs required a more sophisticated approach, including a discourse interpretation module and a segment-based approach to focusing. But although the discourse structure of TFRs forced a number of issues, the fact that this structure is explicit and constant across discourses greatly facilitated the analysis of TFR discourse, to which we now turn.

TFRS AS DISCOURSE

The perspective of a sentence-based grammar might lead us to ignore the formatted lines of a TFR, to consider as discourse only the textual portions, and to interpret each element of the latter as a full or a 'fragmentary' sentence (cf. [Linebarger 88]). On this approach, we would be prepared to analyze the following TFR extract as discourse:

WHEN ATTEMPTING TO ERASE 2 METERS ON THE EVENT RECORDING STRIP, THE STRIP WOULD CONTINOUSLY RUN. INVESTIGATION REVEALED THAT "NOYB" WAS BEING GENERATED. AGE AND USE. REPLACED WITH NEW ITEM AND RETURNED OLD TO SUPPLY. NONE.

However, it is immediately apparent that this approach would be incorrect: the discourse is incoherent.

Two distinct problems may be identified. After the first two sentences, the remainder bear no apparent relation to preceding discourse. Secondly, one or more discourse entities appear to be missing: age and use – of what? Who (or what) replaced what with a new item? None – of what?

The source of incoherency is two-fold: we are missing the initial context established by the interpretation of the formatted lines of the TFR, and we have ignored the basic unit of TFR discourse: the REQUEST-RESPONSE PAIR. As it turns out, each of the elements of the formatted lines (henceforth the *header*) has a positional interpretation, and each of the labels A-E maps to a noun phrase label. Each label can be interpreted as a request for information. Now reconsider the TFR above in this light:

TFR number	**:1234567**
Equipment code	**:TRANSPORTER**
Part number	**:01223426**

[2] As we are not permitted to cite data from actual TFRs, all examples in this paper are purely fictional. However, the crucial linguistic properties have been preserved.

[3] But a recent study has shown them to be even more serious than users of keyword systems might have realized ([Blair 85]).

Date of Trouble :1/21/89
Report date :2/15/89
Originator :JONES

A. First indication of trouble: WHEN ATTEMPTING TO ERASE 2 METERS ON THE EVENT RECORDING STRIP, THE STRIP WOULD CONTINOUSLY RUN.
B. Part failure: INVESTIGATION REVEALED THAT "NOYB" WAS BEING GENERATED.
C. Probable cause: AGE AND USE.
D. Action taken: REPLACED WITH NEW ITEM AND RETURNED OLD TO SUPPLY.
E. Remarks: NONE.

The discourse is now coherent. As can be seen, responses are interpreted relative to their labels, not to each other. The previously missing discourse entity for the referent of NONE is evoked by the label Remarks (i.e., *No remarks*), what was replaced is the failed part (identified by the part number), it is the speaker (JONES) who replaced it, and finally, the implicit argument of AGE AND USE is that same failed part.

These results underline the need to consider the entire TFR as discourse, and to provide an account of the request-response pair as the basic unit of TFR discourse. In the following sections, we sketch such an account, and then turn to the evidence for higher-level structure.

The Request-Response Pair

Between the request and the response a special type of cohesive relation ([Schiffrin 87]) exists, similar to that which binds question-answer pairs. In fact, we claim that at the level of discourse interpretation, the request and response form a discontinuous predicate-argument structure[4]. This view of the request-response pair arises from the need to account for the interpretation of pairs such as Probable cause: BROKEN WIRE, from which we are somehow able to conclude: *The respondant believes that a broken wire caused the failure.*

Very briefly, we suggest that the mechanisms required to achieve this result are essentially those required (at the level of sentence grammar) for the interpretation of specificational copular sentences[5]: lambda-abstraction, function application, and lambda-reduction. First, we take the heads of NP labels to be relational nouns with internal argument structure. For both (1a) and (1b) below, we derive the representation in (2) by lambda-abstracting on the free variable. Function application and lambda-reduction yield the representation in (3), which is (non-coincidentally) also the representation of *A broken wire caused the failure*:

1a. The cause of failure was a broken wire.
1b. Cause of failure: broken wire
2. [λx[cause(x,failure)]](wire)
3. cause(wire,failure)

Discourse Segmentation, Focusing, and Reference

Each label in the TFR marks the start of a request-response pair. But does this unit correspond to a discourse segment, and if so, what is the higher-level structure of the TFR? We studied patterns of reference in TFRs and found evidence for both explicit and implicit structure, as described below.

The Role of the Message Header. The message header identifies the author of the report, the date on which it was sent, the date on which the problem occurred, the equipment, and the failed part. The dates are crucial to the temporal analysis of the message (which we shall not discuss here). Our analysis of the TFR corpus reveals the remaining entities (speaker, equipment, failed part) to be highly salient in the

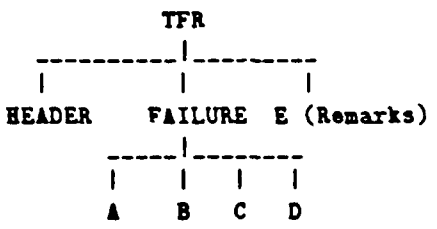
[4]Specifically, we take the NP label to express an OPEN PROPOSITION ([Prince 86]), which can be viewed as an informationally incomplete predication; the response provides its argument.
[5]See for example [Higgins 79] and [Delahunty 82].

discourse: they are available for pronominal reference in segments A-E, without requiring reintroduction by a full NP.

In addition, these entities fill implicit argument positions in the agentless passive, in possible intransitive uses of certain verbs (*replace*, *return*), and in some relational nouns (e.g. *age*, *wear*). These facts lead us to assign these three entities the distinguished status of *global foci*: entities which are always salient in the discourse context at the beginning of each new discourse segment.

Sections A-E. To determine whether each of these sections (`First indication of trouble`, `Part failure`, `Probable cause`, `Action taken`, `Remarks`) constitutes a discourse segment, we studied patterns of pronominal reference in the responses. The results were striking. In 804 occurrences of referential pronouns (707 of which were zero-subjects[6]), we found that only zero-subjects, *I*, *we*, and *this* refer beyond the boundary of the current request-response pair. 90% of the zero-subjects and all of the occurrences of *I* refer to the speaker. The remaining 5% of zero-subjects are distributed between reference to one of the global foci and segment-internal reference, with a slight bias towards the latter. *It*, *he*, *they*, *these*, *those* were found to refer purely locally (*that* did not occur). With the exception of *this* and the indexicals, pronominal reference is sensitive to the boundary of the request-response pair, and we conclude that each such pair is indeed a discourse segment.

In the demonstrative *this*, however, we found unexpected evidence for additional *implicit* structure: when occurring in segment E (`Remarks`), *this* can refer to the failure, or problem, described in segments A-D. Now, [Webber 88] argues that demonstrative reference of this type is sensitive to the *right frontier* of the discourse tree: that is, 'the set of nodes comprising the most recent closed segment and all currently open segments' (Webber 1988:114). If, as we had assumed, segments A-E are sisters, then segment D (`Action taken`) is the most recently closed segment, and there are no segments open other than the current segment, E. But none of the occurrences of *this* in segment E refer to segment D. To make sense of the data, we were led to the conclusion that segments A-D form an unlabelled, implicit segment: *the failure*. The `Remarks` segment is then the sister of this implicit segment; after closing segment D, this higher segment is closed, and thus lies on the right frontier when E is opened. From these observations we posit the following structure for the TFR:



THE TFR APPLICATION

The TFR application uses the PUNDIT natural-language processing system to analyze TFRs. The results of analysis are passed to a database module, which maps PUNDIT's representations to pre-defined records in a Prolog relational database. This database can then be queried using a natural-language query facility (QFE). Here, we discuss only the analysis part of the application.

In terms of user interaction, the TFR data-collection program superficially resembles traditional data-processing approaches to forms automation: the system prompts for each item on the form, and the user's response to each prompt is validated. If the response is judged invalid, an error message is issued and the user is reprompted.

[6] As in INSTALLED NEW ITEM, RETURNED OLD TO SUPPLY.

Under the covers, however, the approach is quite different: the data-collection program is in fact a discourse manager, controlling and interpreting a dialogue between itself and the user. As the dialogue proceeds, it maintains a model of the discourse, calls PUNDIT's syntactic and semantic/pragmatic components to analyze the user's responses, and then interprets the response in the context of the prompt to derive new propositions. In addition, it manages the availability of discourse entities, moving entities in and out of focus as the discourse proceeds from one segment to the next.

IMPLEMENTATION

The TFR Discourse Manager is implemented as a single top-level control module, written in Prolog, which uses PUNDIT as a resource. Its highest-level goals are to collect pre-defined information from the user and send the resulting information state to a database update module.

At the level of user interaction, the module's goals are to process the request-response units corresponding to the header items and the segments A-E. In the header segment, the Discourse Manager prompts for each of the header items (speaker, date, part number, etc.), and calls PUNDIT to analyze the responses. The responses give rise to discourse entities, whose representations are added to the DISCOURSE LIST for subsequent full-NP reference. The three global foci (speaker, failed part, and equipment) are stored in a distinguished location in the discourse model.

For each of the remaining segments (A-E), the processing is described below.

1. Initialize Discourse Context

   At the start of each segment, we empty the list of salient entities from the previous segment (the FOCUS LIST) and load in the global foci. This prevents pronominal reference from crossing segment boundaries (although full NP reference is possible).

2. Prompt the User

   Before the system can interpret the user's response to a prompt, it must first 'understand' what it is about to ask. This step, while intuitive, is actually required in order to create the context for interpreting the response. We look up the meaning of the prompt (stored as a lambda expression), create a discourse entity, and place it at the head of the focus list. This makes the prompt the most salient entity in the context when the response is processed, and allows for both pronominal and implicit reference, e.g. **Probable cause: UNKNOWN**. Having done this, we issue the prompt and collect the user's response.

3. Analyze the Response

   Two levels of interpretation are provided. First, PUNDIT is called to analyze the response; next, the response entity is bound to a variable in the representation of the prompt, to derive a new proposition.

   Two types of call to PUNDIT are required, in order to handle both NP responses (**BROKEN WIRE**) and sentential or paragraph responses (**BELIEVE PROBLEM TO HAVE BEEN CAUSED BY FAILURE OF UPPER WIDGET**). If the response can be analyzed by PUNDIT's syntactic component as an NP, then a side-door to PUNDIT semantic and pragmatic analysis is used to provide a semantic interpretation and create a discourse entity.

   If the response cannot be analyzed as an NP, then the normal entrance points for syntactic and semantic/pragmatic analysis are used. This results in the creation of one or more situation entities, which are grouped together to form a higher-level response entity.

   Finally, the response entity is bound to the variable in the representation of the prompt, and lambda reduction is applied. The resulting representation is added to the discourse list, where it becomes available for subsequent full-NP reference (e.g. *The failure...*, *The cause...*).

RESEARCH DIRECTIONS

The implementation described above partially captures our observations concerning the discourse structure of TFRs and how it constrains pronominal reference, as well as the discourse relation of requests to responses. It thus provides a level of discourse management and interpretation beyond that developed for previous PUNDIT applications. Our experience with this application has led us in two research directions: towards the management of open-ended dialogue, and towards the development of a domain-independent discourse interpretation facility.

References

[Blair 85] Blair, David C. and Marion, M. E. An Evaluation of Retrieval Effectiveness for a Full-Text Document Retrieval System. *Communications of the ACM* 28(3):289–299, 1985.

[Dahl 86] Dahl, Deborah A. Focusing and Reference Resolution in PUNDIT. In *Proceedings of the 5th National Conference on Artificial Intelligence*. Philadelphia, PA, August 1986.

[Dahl 87] Dahl, Deborah A., Dowding, John, Hirschman, Lynette, Lang, Francois, Linebarger, Marcia, Palmer, Martha, Passonneau, Rebecca, and Riley, Leslie. *Integrating Syntax, Semantics, and Discourse: DARPA Natural Language Understanding Program, R and D Status Report*. Technical Report, Unisys Corporation, May 1987.

[Delahunty 82] Delahunty, Gerald P. *Topics in the Syntax and Semantics of English Cleft Sentences*. Indiana University Linguistics Club, Bloomington, 1982.

[Grishman 86a] Grishman, Ralph and Hirschman, Lynette. PROTEUS and PUNDIT: Research in Text Understanding. *Computational Linguistics* 12(2):141–45, 1986.

[Grishman 86b] Grishman, Ralph and Kitteridge, Richard (editors). *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*. Lawrence Erlbaum, New Jersey, 1986.

[Grosz 86] Grosz, Barbara J. and Sidner, Candace L. Attention, Intentions and the Structure of Discourse. *Computational Linguistics* , 1986.

[Higgins 79] Higgins, F. R. *The Pseudo-Cleft Construction in English*. Garland, 1979.

[Hirschberg 87] Hirschberg, Julia and Litman, Diane. Now Let's Talk about *now*: Identifying Cue Phrases Intonationally. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 163–171. Stanford, CA, July 1987.

[Linebarger 88] Linebarger, Marcia C., Dahl, Deborah A., Hirschman, Lynette, and Passonneau, Rebecca J. Sentence Fragments Regular Structures. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*. Buffalo, NY, June 1988.

[Prince 86] Prince, Ellen F. On the Syntactic Marking of Presupposed Open Propositions. In *Papers from the Parasession on Pragmatics and Grammatical Theory at the 22nd Regional Meeting of the Chicago Linguistic Society*. 1986.

[Schiffrin 87] Schiffrin, Deborah. *Discourse Markers*. Cambridge University Press, Cambridge, 1987.

[Webber 88] Webber, Bonnie Lynn. Discourse deixis: Reference to Discourse Segments. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 113–122. Buffalo, NY, June 1988.

Dahl, D., and Ball, C. "Reference Resolution in PUNDIT", to appear in *Logic and Logic Grammars for Language Processing*, edited by Stan Szpakowics and Patrick St. Dizier, Ellis Horwood, LTD.

Reference Resolution in PUNDIT

Deborah Dahl, Catherine N. Ball

September 18, 1989

Contents

To Appear in Logic and Logic Grammars for Language Processing, Patrick Saint-Dizier and Stan Szpakowicz, eds. Ellis Horwood

1 Introduction

This paper discusses the reference resolution component that has been implemented in Pundit, a large natural language processing system in Prolog. Pundit is currently under development at the Paoli Research Center of Unisys [Hirschman 89]. The reference resolution component has been implemented with three goals in mind. First, it is a system module in Pundit, and as such it contributes to the overall goal of understanding messages, specifically, by resolving references. Second, and more generally, it is intended to provide a test of current theoretical approaches to discourse processing through an implementation as part of a large natural language processing system which is designed to process real texts. Finally, it also serves as a tool on which to base the development of new approaches to problems in reference resolution. In practice, these goals are often intertwined; for example, a text that Pundit is expected to process will often present difficulties that can only be resolved through basic extensions of its algorithms.

In this paper, we will first provide an overview of PUNDIT, situate the reference resolution component with respect to the rest of the system. In Section 3, we will describe the relationship with the semantic interpreter ([Palmer 85]) in more detail and describe the data structures used by reference resolution. In Section 4, we will discuss the two major functions of reference resolution, management of the discourse context and resolving referents. We will conclude with a discussion of areas which are currently inadequately dealt with and future directions.

The problem that reference resolution is aimed at solving is essentially to determine what individuals and situations are being discussed in a stretch of language. When is a new entity brought into the discussion, when is an old one mentioned again, and how are newly mentioned entities related to what has already been discussed? The approach described in this paper is built on a few basic concepts. DISCOURSE ENTITY is a term introduced by Webber ([Webber 78]). and refers to things that can be talked about in a discourse; individuals, situations, actions, material, types, and so on. It is essentially the same idea as Kartunnen's (Kartunnen 1969) **discourse referents.**, or Heim's ([Heim 82]) 'file cards'. The second basic idea that this work depends on is the idea of INFORMATION STATUS, which refers to the cognitive accessibility of discourse entities. A very large number of information statuses have been proposed in the literature, but we will make particular use here of the notion of focusing/centering ([Sidner 79], [Grosz 86],[Grosz 83]).

2 Overall Organization of PUNDIT

The overall organization of PUNDIT can be seen in Figure 1. The input first goes through a syntactic analysis phase where it is parsed and regularized (Hirschman and Dowding, this volume). The regularized parse tree is passed to the semantic interpreter, ([Palmer 85]).

Semantics, reference resolution, and the time component ([Passonneau 88]) interact to produce a final linguistic representation, the INTEGRATED DISCOURSE REPRESENTATION, or IDR,which then serves as the input to application modules. Application modules have been built to produce database updates, database queries, and tabular summaries, all using the same IDR. A forward chaining inference mechanism has recently been added to supply a capability for non-linguistic reasoning.

3 Reference Processing

3.1 Roles of Reference Resolution

Within the PUNDIT system, the reference resolution component plays several roles. Its primary function is to supply referents both for explicit referring expressions and discourse entities whose existence is implied only indirectly by the text. In order to perform this function, the reference resolution component also maintains a discourse model (referred to as the DISCOURSELIST,) which represents known discourse entities, their properties, and the situations in which they participate. A third function is to maintain a representation which indicates which discourse entities are available for pronominal reference and in what order they should be considered as referents for pronouns.

In this section we describe the relationship between the semantic interpreter and reference resolution in more detail, we describe the data structures currently created and used by the reference resolution component, and conclude with a discussion of how logic variables are used to represent discourse entities.

3.2 Interaction with the Semantic Interpreter

The interaction between reference resolution and the interpreter for clause level semantics ([Palmer 85]), is quite complex. The semantic interpreter interprets rules which define the semantic roles and semantic decomposition for each predicate. It looks at each role in succession, and attempts to fill the role with a discourse entity which is associated with a syntactic constituent, according to rules specifying how roles and constituents are associated. Thus, when the semantic interpreter is ready to fill a role, it calls a component which provides a semantic and pragmatic analysis for the constituent associated with the hypothesized role filler. If the constituent is a noun phrase (it can also be a clause, for example, if the verb takes clausal complements), the semantic interpreter calls a component for noun phrase analysis. The noun phrase analysis component calls both noun phrase semantics and reference resolution. Noun phrase semantics interprets the noun phrase and then reference resolution finds the referent of the noun phrase. It returns a constant which represents the referent. Selectional restrictions on the role are checked by the semantic interpreter, and if the referent is acceptable, it then becomes the filler of the semantic role. The semantic interpreter

then proceeds to fill other roles. If the referent is not acceptable, then backtracking into the reference resolution component will occur, and another referent will be proposed. Complications arise in certain complex constructions, in particular nominalizations. Although they are noun phrases syntactically, nominalizations such as *failure* have a semantic interpretation which is very similar to that of the related verb, e.g. *fail*, and therefore it makes sense to use the same semantic interpreter on both *fail* and *failure*. Thus, when noun phrase semantics determines that the head of the current noun phrase is a nominalization, it calls the clause level semantic interpreter to interpret that noun phrase. A detailed discussion of this interaction can be found in ([Dahl 87]). The time component is called at the end of clauses and other constituents requiring temporal analysis, such as nominalizations. It produces a situation representation and temporal orderings as discussed in [Passonneau 88].

3.3 Data Structures

The reference resolution component maintains several data structures which function to make the discourse context available throughout processing. LOCALFOCI is a list of semantically and pragmatically relevant information from the current sentence, both from the current clause and from other, previously analyzed clauses. The information from each clause is maintained in a separate sublist. The DISCOURSELIST is a list of pragmatically relevant information from previous sentences of the text. Finally, the FOCUSLIST is a list of discourse entities that are eligible for pronominal or elided reference, as well as other functions described in Section 4.

3.4 Logic Variables as Discourse Entities

Logic variables have provided an interesting tool for the implementation of referential indices and discourse entities. Referential indices are traditionally used in linguistic description to indicate syntactic constraints on anaphoric binding. In PUNDIT'S syntactic processing, each noun is associated with a logic variable which represents its referential index. Certain control phenomena, known in the linguistics literature as equi and raising, are then handled in the syntax by unification of the logic variables representing the referential indices of the filler and the gap. For example, in the sentence, *The field engineer attempted to repair the pump*, the referent of gap in the subject position of *repair* is unified with the referential index of *the field engineer* so that when *the field engineer* is resolved the subject of *repair* is determined simultaneously. Similarly, gaps in relative clauses receive the same referential index as the head of the clause, as in *The field engineer who repaired the pump replaced the motor*, and resolving the referent of the head automatically results in instantiating the referent of the gap. This strategy is attractive because it implements the linguistics of referential indices and coreference relationships in general in a very declarative and transparent way, by unifying variables and also allows us to express variables bound by quantifiers very directly.

On the other hand, this implementation has limitations because it precludes PUNDIT from being able to assert data structures containing the uninstantiated variables and still maintain the desired variable bindings. For this reason, as well as to enhance modularity, all of the discourse data structures described above are not asserted, but are passed around the system in a parameter.

3.5 Introduction

PUNDIT divides the task of reference resolution into two tasks, managing the discourse context, specifically, integrating new information into the data structures that represent the discourse, and actually resolving the referents of noun phrases and implicit and elided entities. We begin this section by discussing some issues raised by message processing–in particular, the telegraphic style typical of messages. We then discuss the issues surrounding discourse context management, in particular, the uses of focusing, and the problem of preferred focus order. Finally, we discuss the specific reference processing algorithms used in PUNDIT for the different types of referring expressions.

3.6 Message Processing

In our applications to date, we have used PUNDIT to analyze a variety of short military messages: CASREPs (equipment failure reports), RAINFORMs (tactical messages about ship sightings), TFRs (equipment Trouble and Failure Reports) and OPREPS (messages about naval engagements). These messages have several characteristics make them interesting from the point of view of reference resolution: they have varying degrees of *explicit* discourse structure, and they are written in what is often called *telegraphic style*. The following Trouble and Failure Report illustrates both of these points.

```
<Formatted lines...>

A. WHILE PERFORMING SDC 955Z (GENERATION OF LASER BEAMS) UPPER TRANSLOCK
WENT OFF LINE. B. UPPER TRANSLOCK INTERPORT SWITCH WENT BAD, UNABLE TO
RE-ENERGIZE ETHER REGULATOR IN UPPER TRANSLOCK WHEN INTERPORT SWITCH
DEPRESSED. C. DETERIORATION DUE TO AGE AND WEAR. D. REPLACED INTERPORT
SWITCH WITH A NEW ONE FROM SUPPLY. E. NONE.
```

Discourse Structure

Notice that the message is composed of formatted lines (the *message header* and a message body. Within the message body, the labels A-E have a fixed interpretation: the A section describes the first indication of trouble, B the failure of a part, C the probable cause of failure, D the action taken, and section E is the Remarks section. As we shall discuss, each section corresponds to a *discourse segment*, and the segmentation of the discourse crucially constrains pronominal reference: it is not in general possible for a pronoun to refer beyond the boundaries of the current segment.

It turns out that it is also crucial to take into account the formatted lines of the message header. In military messages, the header typically identifies the message originator, the date/time on which the message was sent, perhaps the date/time on which the main event occurred (in this case, the date of the equipment failure), and other information which is relevant to the interpretation of the message. In short, the header creates the initial discourse context for the message body. How PUNDIT uses this information is discussed in detail in [Ball 89].

Telegraphic Style

When a message such as that above is compared to standard written English, it is apparent that several types of linguistic expression appear to have been omitted. For example, the definite article *the* does not occur in the message (as in `[the] UPPER TRANSLOCK WENT BAD`). There is no overt subject for the sentence `[I] REPLACED INTERPORT SWITCH WITH A NEW ONE FROM SUPPLY`. The copula is missing in `...WHEN INTERPORT SWITCH [was] DEPRESSED`.

As we observe in [Linebarger 88], these omissions are not random, but are highly regular. While the omission of determiners and subjects might be thought to make the result highly ambiguous, in fact there are preferred interpretations in such cases: for example, an algorithm that first tries to construe a determinerless noun phrase (`UPPER TRANSLOCK`) as definite (*the*) yields good results. Similarly, we found that approximately 95% of the zero-subjects in our corpus of messages refer to the message originator (*I*. Therefore it makes sense to first try to interpret zero-subjects as referring to the message originator; only if this strategy fails should we try other candidates from the context (as in `HAPPENS ONLY INTERMITTENTLY`, where the verb is third person).

Lest it be thought that messages represent an unusual dialect of English, and that the phenomena we will be discussing are rather peripheral, we should point out that the same types of phenomena can be observed in casual speech, as well as in other genres of written language. For example, consider the zero-subjects in *Gotta run*, and *Looks like rain*. In newspaper headlines, signs, answers on forms, notes, postcards, cookbooks, directions on packages, and other genres of writing we can also find the telegraphic style at work: *Reagan Tied to Aid Pact, Having a great time, wish you were here, Cook until done, Close cover before striking*, etc. It is clear that the ability to produce and interpret such utterances is part of our competence as speakers of English. In message processing, we are forced to confront the issue of how these are understood.

3.7 Local Focusing

3.7.1 Local Focusing

Local focusing is largely concerned with relationships of discourse entities from sentence to sentence, as opposed to global focus, which is concerned with a text or discourse as a whole. The idea of focusing, as developed by researchers such as ([Grosz 77], [Sidner 79]) as well

as the closely related concept of centering, as discussed by ([Grosz 83] and [Grosz 86]) has been used in natural language systems for the purpose of resolution of definite references ([Dahl 86], [Brennan 87]). The basic idea is that certain entities in a discourse will be established as being 'on the minds of' the speaker and hearer in such a way that some form of pronominal or other reduced form of reference to those entities will be felicitous. This status has been variously referred to in the linguistics literature as 'given' (Chafe 1976), or 'activated' (Gundel 1978).

'Focus' as used by [Grosz 77] is distinct from the term as used by Sidner. Grosz's concept is more relevant to a discourse entity that is given throughout an entire discourse, while Sidner's concept, and centering apply to an entity that is more locally given. We will refer to the first concept as 'global focus' and the second concept as 'local focus'. Pundit makes use of both mechanisms. In this section, we discuss local focus. PUNDIT'S use of global focus is discussed in [Ball 89].

Linguistically reduced forms, such as pronouns, are used to refer to the entity or entities with which a discourse is most centrally concerned. Keeping track of this entity, which corresponds to (the *topic* of [Gundel 74], the *focus* of [Sidner 79] and the *backward-looking center* of ([Grosz 86], [Grosz 83] and [Brennan 87]) has typically been used in the interpretation of pronouns. However, while 'pronoun resolution' is generally presented as a problem in computational linguistics to which focusing can provide an answer (see for example, the discussion in (Hirst 1981)), it is useful to consider focusing as a problem in its own right. By looking at focusing from this perspective, it can be seen that focusing plays a role in the interpretation of several types of noun phrases. In PUNDIT, for example, local focusing is used in the interpretation of a variety of forms of anaphoric reference; in particular, pronouns, elided noun phrases, *one*-anaphora, and context-dependent full noun phrase references [Dahl 86], (Ward 1985 also uses centering for constraining left-dislocation).

3.8 Focus Order

In Sidner's implementation of focusing and in many later discussions and implementations, the notion of 'focus order'or the preferred ordering of forward looking centers has been used as a means of allowing the system to prefer certain discourse entities as pronominal referents in the following sentence (always allowing for the possibility that semantic constraints will rule out the preferred entity). Within the focusing/centering literature, it has generally been assumed that there is a single, domain independent mechanism for determining focus order. For example, Sidner's algorithm, which is based on thematic roles, prefers the theme of the previous sentence as the focus of the current sentence. PUNDIT in its current implementation considers the entire previous utterance to be the preferred potential focus In the centering literature, the subject is generally considered to be preferred. Each of these approaches also characterizes a list of less and less preferred potential foci. There have been very few direct arguments addressing the reasons for this discrepancy among orderings. It is our belief that the reasons behind these differences are probably due to both insufficient data and possible

effects of the type of discourses being dealt with in different systems. Because of this, we want to make the basis of focusing easy to change and easy to experiment with. For this reason, we have implemented a focus order interpreter in PUNDIT. This interpreter orders potential foci according to a simple declarative specification that is easily changed for the purposes of experimentation. Currently, for example, the specification is:

focus_order([current_event,pronoun,object,subject,sentence_pps]).

It is also possible to define a 'pruning value' which cuts off the list of potential foci after any desired length. In order to experiment with alternate orderings, it is only necessary to rearrange the elements in this list. If any additional definitions of potential foci are of interest, it would be necessary to define a procedure for finding them. For example, if it was of interest to implement a version of Sidner's role-based focusing, it would be necessary to implement procedures for finding the theme, actor, patient, and other thematic roles. We have experimented with the subject as the preferred focus in some domains and have found that it is more frequent than object, and hence that if the subject is ordered before the objects the processing is more efficient, however we have not yet determined whether referential errors will result.

3.9 Reference Resolution

A number of different types of referring expressions are handled by the reference resolution component in PUNDIT. These include definite and indefinite noun phrases, as well as noun phrases without determiners; pronouns and elided noun phrases; and certain types of one-anaphora. The system also handles references to situations as well as references to objects. While reference resolution is typically thought of with respect to noun phrases, other constructions, clauses in particular, display properties which resemble the referential properties of noun phrases. For example, clauses can express 'old' or 'given' information in the same way that noun phrases can refer to a previously mentioned discourse entity.

The discussion of reference resolution is divided into three sections. The first section describes our treatment of full noun phrases, the second is concerned with the uses of focusing to handle various cases of reduced reference, such as pronouns, elided noun phrases, and *one*-anaphora, and the third section describes an experimental approach to handling clauses.

3.9.1 Full Noun Phrases

The definite determiner *the* typically signals that a referent has been previously mentioned or is otherwise known, so that *the pump* is taken to refer to some known pump. Conversely, *a pump* is taken to refer to a pump that is not previously known to the reader. This distinction can be used by a reference resolution component to decide whether or not the noun phrase refers to a new entity. However, in the telegraphic style of speech characteristic of most of the domains we have analyzed, determiners are nearly always omitted. Their

function must therefore be replaced by other mechanisms. One possible approach to this problem might be to have the system try to determine what the determiner would have been, had there been one, insert it, and then resume processing as if the determiner had been there all along. This approach was rejected for two reasons. The first is that it was judged to be more error-prone than simply equipping the reference resolution component with the ability to handle noun phrases without determiners directly. The second reason for not selecting this approach is that this would eliminate the distinction between noun phrases which originally had a determiner and those which did not. Since this distinction may have a discourse function, we wanted to retain it.

The current approach deals with this issue as follows. If a noun phrase has an explicit indefinite determiner (and the noun is not *dependent* , in the sense discussed in the next section), the noun phrase is assumed to refer to something not previously known, and a new unique identifier is created for that referent. In contrast, all other noun phrases trigger a search through the discourse context for a previously mentioned referent. Both noun phrases with an explicit definite determiner, and those with no determiner are handled in this way. This amounts to making an initial assumption that noun phrases without determiners are definite, but confirming this with a search through the context. If the search for a previously mentioned referent fails, then another search is initiated which determines if the referent of the noun phrase is related to some other referent in focus (that is, whether the two referents are *implicit associates* as discussed below). If this fails, then a new unique identifier is created, just as if the noun phrase had been indefinite.

In order to determine whether a referent has been previously mentioned, the properties in the current noun phrase are matched against the properties of previous noun phrases. As long as all of the new properties are also properties of the old referent, the referents match. So in a discourse such as *New pump arrived. Pump was installed.* the second time the pump is mentioned, it does not again have to be referred to as the new pump. It is also possible to refer to an entity for the second time using a superordinate term, as in *Pump failed. The machine has been failing repeatedly.* PUNDIT is able to recognize that *machine* is a reference to the pump by looking for a subordinate concept in the previous discourse.

3.9.2 Uses of Focusing

A focusing algorithm based on surface syntactic constituents is used in the processing of several different types of reduced reference: definite pronouns, *one-* anaphora, elided noun phrases, and implicit associates. The focusing mechanism in this system consists of two parts–a FOCUSLIST , which is a list of entities in the order in which they are to be considered as foci, and a focusing algorithm, which orders the FOCUSLIST using the focus order interpreter described above. A detailed description of the focusing mechanism is available in [Dahl 86].

Pronoun resolution is done by instantiating the referent of the pronoun to the first member of the FOCUSLIST. Selectional restrictions on referents are checked after reference resolution

is exited. Backtracking into reference resolution occurs if selection fails and then the next member of the FOCUSLIST is proposed as a referent. If all focused entities are rejected, a domain-specific default is used.

The reference resolution situation in message texts is, however, complicated by the fact that there are very few overt pronouns. Rather, in contexts where a noun phrase would be expected, there is often elision, or a zero-NP as in *Received new pump* and *Investigated failure* . Zeroes are handled basically as if they were pronouns [Linebarger 88]. Our analysis of the texts has indicated that the most frequent, although not the only, interpretation of zero is as a first person reference. For this reason, PUNDIT first attempts to interpret zeroes as first person references, but then uses other discourse entities in focus if the first person reference fails. The hypothesis that elided noun phrases can be treated in the same way as pronouns is consistent with previous claims in ([Gundel 80], Kameyama 1985) that in languages such as Russian and Japanese, which regularly allow zero-NP's, the zero corresponds to the focus. If these claims are correct, it is not surprising that in a sublanguage like that found in the maintenance texts, which also allows zero-NP's, the zero can also correspond to the focus.

Another complication in the interpretation of anaphoric pronouns and elision arises when the anaphor occurs as the subject of a subordinate clause. In these cases there is a preference for the anaphor to be interpreted as coreferential with the subject of the main clause, as in *The original drive shaft, when installed, was packed utilizing 60 grams of grease.* where the subject of *install* is clearly the drive shaft. PUNDIT is sensitive to the syntactic context of the anaphor it is interpreting so as to capture this preference. This also allows PUNDIT to correctly interpret the class of forward anaphora where the anaphor is in a sentence modifier, as in the above example. Because PUNDIT processes main clauses before adjuncts, the pragmatic information required for reference resolution will be available when the elided subject of the adjunct is processed, whether the adjunct precedes or follows the main clause. This order of processing also (correctly) prevents full noun phrases in adjunct clauses from being interpreted as the antecedents of anaphors in the main clause, because they will not yet have been processed when the anaphor is interpreted. Finally, our implementation allows a referent from the discourse to fill in the elided subject when the subject of the main clause is semantically unacceptable. Future work includes extension of this treatment to lexical pronouns, and other subordinate clauses in addition to sentence modifiers.

In addition to its uses in interpreting pronouns and elided subjects, focusing is also used in the processing of certain full noun phrases, both definite and indefinite, which involve *implicit associates.* The term implicit associates refers to the relationship between *a disk drive* and *the motor* in examples like *The field engineer installed a disk drive. The motor failed.* It is natural for a human reader to infer that the motor is part of the disk drive. In order to capture this intuition, it is necessary for the system to relate the motor to the disk drive of which it is part. Relationships of this kind have been extensively discussed in the literature on definite reference. For example, implicit associates correspond to *inferrable* entities described in [Prince 81], the *associated use definites* of [Hawkins 78], and the *associated* type of implicit backwards specification discussed in [Sidner 79]. Sidner suggests that implicit

associates should be found among the entities in focus. Thus, following this suggestion, when PUNDITencounters a definite noun phrase mentioned for the first time, it examines the entities in focus to determine if one of them is a possible associate of the current noun phrase. The specific association relationships (such as **part-whole** , **object-property** , and so on) are defined in the knowledge base. Hawkins (in press) specifically hypothesizes that association relationships are the same as slot-frame relationships in a frame-based knowledge representation system, and we have adopted this assumption in PUNDIT. Thus, correctly dealing with these relationships requires interaction with a knowledge base.

This approach is also used in the processing of certain indefinite noun phrases. In every domain, there are certain types of entities which can be classified as *dependent* . By this is meant an entity which is not typically mentioned on its own, but which is referred to in connection with another entity, on which it is dependent. For example, in the starting air compressor domain, *oil* is classified as such a dependent entity, since it is normally mentioned with reference to something else, such as a *starting air compressor* . When an indefinite dependent noun phrase is encountered, PUNDIT attempts to relate it through an association relationship to an entity in focus.

Finally, PUNDIT extends focusing to the analysis of *one-* anaphora following [Dahl 84] which claim that focus is central to the interpretation of *one-* anaphora. Specifically, the referent of a *one-* anaphoric noun phrase (e.g., *the new one, some defective ones*) is claimed to be a member or members of a set which is the focus of the current clause. For example, in *Three sacs installed. One failed* , the set of three sacs is assumed to be the focus of *One failed* , and the sac that failed is a member of that set. A similar approach is advocated in [Webber 83]. The main computational advantage of treating *one-* anaphora as a discourse problem is that the basic anaphora mechanism then requires little modification in order to handle *one-* anaphora. Alternatively, treating *one* -anaphora as a purely syntactic phenomenon (as suggested by [Halliday 76] and [Webber 78]) would require an entirely separate mechanism.

Finally, as mentioned above, reference resolution is also relevant to clause interpretation. Every clause is associated with a discourse referent, which is intended to correspond roughly to a Davidsonian event variable (Davidson 1967). These referents are available in the discourse context for reference through pronouns and full definite noun phrases, just as are the referents evoked by noun phrases (example). In addition, clauses that occur later in the discourse can also refer to the same situation as an earlier clause. Normally, the subsequent reference is marked in some way as presupposed; for example, it is a relative clause or other type of subordinate clause. For example,

The Lexington visited Philadelphia. When the Lexington visited Philadelphia, the Enterprise was also in port.

Clearly, there is only one visit of the Lexington to Philadelphia under discussion here. The process of associating discourse referents with clauses is then very similar to that of finding referents to noun phrases. After the semantic processing for a clause is complete, the system looks through the context for a previously evoked situation which contains the same information. If there is one, then the two situations are assumed to be the same. This treatment

of clauses is experimental and requires confirmation by additional data; however, it seems promising, and it is clear that some mechanism of this sort will be required to correctly handle clauses.

4 Future Directions

Although PUNDIT'S reference resolution component covers a wide variety of cases, it is far from complete. We discuss here a few interesting examples of areas where it falls short.

- Superordinates

  For ordinary entities, PUNDIT can handle reference by a superordinate term. For example, in the RAINFORMs domain model the Admiral Golovko is defined as a Kynda-class cruiser, and cruisers are ships; hence the Golovko can be referred to by the terms *ship*, *cruiser*, or *Kynda*, as in (2-57) [visual detection of **Admiral Golovko**] EXCHANGE MISSILE FIRE WITH **KYNDA**.

  For situations, we have at present no similar capability (although it is a planned extension). Thus PUNDIT is unable to recognize that an attack with guns can be referred to as an engagement, e.g. as in (2-27) *FANNING AND MILLER... CONDUCTED GUN ATTACKS ON KOBCHIK. FANNING AND MILLER HAVE BROKEN ENGAGEMENT.* There are numerous examples of reference to situations by superordinate terms in the RAINFORM corpus, and this represents a significant coverage gap.

  Similarly, for full plural noun phrases (e.g. *the ships*), there is provision for only one type of reference: reference to ordinary entities (non-situations) which are referred to in essentially the same way that they were first introduced. E.g. *Sighted a ship. Sighted another ship. The ships....* We cannot yet handle reference by a superordinate term (e.g. *Sighted Barsuk; sighted Peleng; attacked* **the ships**.

- Plural Pronouns and Nominalizations

  PUNDIT has no provision for resolving the reference of plural pronouns (*we*, *they*). These are, however, exceedingly rare in the corpora we have dealt with (as are all overt pronouns). Because the facts about plural pronouns are quite complex (see, for example, the discussion in [Sidner 79]) we have delayed implementing a treatment of them until we have a domain with enough examples to ensure generality.

  The reference resolution component is able to correctly resolve the reference of plural nominalizations (*the attacks*); however, this treatment is incomplete, because correctly representing plural situations is quite complex. For example, in a noun phrase such as *two torpedo attacks* PUNDIT represents the instrument of both attacks as the same torpedo, which is clearly incorrect. In order to handle this situation properly, it seems that domain specific reasoning will have to be involved. We know, for example, that

projectiles used in an attack are in a sense 'used up' after one attack, and this knowledge should force the system to represent the attacks as involving distinct sets of torpedoes. On the other hand, there are cases where the same argument should be assigned to multiple situations, or whether the identity of the argument is vague. A more sophisticated treatment would be able to deal correctly with the arguments of such nominalizations. In addition, a much more sophisticated component for temporal processing would be required in order to properly capture the temporal relationships among the situations. For example, the relationship in *two torpedo attacks* seems to be sequential, while in *several pump failures,* the failures could be simultaneous.

- Non-referential Nominalizations

  A very simple treatment of non-referential nominalizations has been implemented, but as in the case of plural references to situations, a great deal of complexity remains to be dealt with. It is important to represent non-referential noun phrases such as *no failure* such a way that the system does not behave as if a failure had occurred. For noun phrases with the determiner *no*, a representation is created which indicates that there is no specific individual of the type of the head noun in the discourse context. However, it would be incorrect to simply ignore the noun phrase altogether, because it does have consequences for later reference. In our treatment, the generic idea of *failure* is represented in the discourse context, because it is normally available for further reference, as in the hypothetical example, *No failures at this time. None expected.* Because the interpretation of *none* is clearly 'no failures', the generic idea of 'failure' is represented in the discourse context. However, a complete treatment of this phenomenon would be quite complex. For example, it seems likely that some, but not all the arguments of a non-referential nominalization should be non-referential. For example, in *no pump failures* it would not be correct to introduce a new pump into the discourse context. This situation is parallel to the better-known problems that have been discussed in the literature on specific and non-specific indefinites in the scope of negation.

  Part of the reason for the incompleteness of our treatment of non-referential nominalizations is that the treatment of non-referential noun phrases in general is quite superficial. In the texts, (usually reports) we have dealt with, most of the individuals and situations mentioned are 'real'. There are very few hypothetical or counterfactual situations or entities. As a consequence, both the referential and temporal processing in PUNDIT are strongly oriented toward real situations and individuals. The extent to which this orientation will interfere with extending the system to cover hypothetical and counterfactual situations and entities remains to be seen.

- Sense and Reference

  Where the relationship between sense and reference is not direct, PUNDIT is not currently able to provide the correct results. For example, consider the different senses of *MAD* (magnetic anomaly detector) in the following text:

(1-25) GAINED MAD CONTACT AT 2210Z AND ATTACKED ON MAD

In the second clause, *MAD* is used in an extended sense to refer to the object which was detected by MAD (a submarine). PUNDIT is not yet equipped to deal with such cases, which require coercion. In this case, the reference resolution component would simply find the previously mentioned MAD, and the resulting interpretation would be that the ship attacked its own sensor. Furthermore, the semantic interpreter would also reject the MAD as a semantically appropriate argument of *attack*. These example await future research.

- Reference Resolution in a Spoken Language System

  We are presently beginning to extend the PUNDIT system to deal with spoken input. One very promising avenue of exploration will be the possibility of using prosodic information in processing references. There is some evidence that humans use prosodic information in pronoun resolution. For example [Dahl 82] showed that unstressed pronouns are processed more quickly than stressed pronouns, and stressed pronouns are processed differently depending on the function of the stress – it will be exciting to begin to extend these kinds of results to machine processing.

References

[Ball 89] Ball, Catherine N. Analyzing Explicitly-Structured Discourse in a Limited Domain: Trouble and Failure Reports. In *Proceedings of the DARPA Speech and Natural Language Workshop.* Philadelphia, PA, February 1989.

[Brennan 87] Brennan, Susan E., Friedman, Marilyn W., and Pollard, Carl J. A centering approach to pronouns. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 155–162. Stanford, CA, 1987.

[Dahl 82] Dahl, Deborah A. and Gundel, Jeanette K. *Identifying Referents for Two Kinds of Pronouns*, pages 10–29. Volume 8. University of Minnesota, 1982.

[Dahl 84] Dahl, Deborah A. *The Structure and Function of One-Anaphora in English.* PhD thesis, University of Minnesota, 1984.

[Dahl 86] Dahl, Deborah A. Focusing and Reference Resolution in PUNDIT. In *Proceedings of the 5th National Conference on Artificial Intelligence.* Philadelphia, PA, August 1986.

[Dahl 87] Dahl, Deborah A., Palmer, Martha S., and Passonneau, Rebecca J. Nominalizations in PUNDIT. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.* Stanford University, Stanford, CA, July 1987.

[Grosz 77] Grosz, Barbara J. *The Representation and Use of Focus in Dialogue Understanding.* PhD thesis, University of California, Berkeley, 1977.

[Grosz 83] Grosz, Barbara, Joshi, Aravind K., and Weinstein, Scott. Providing a Unified Account of Definite Noun Phrases in Discourse. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics* 44–50, 1983.

[Grosz 86] Grosz, Barbara J., Joshi, Aravind K., and Weinstein, Scott. Towards a computational theory of discourse interpretation. 1986. unpublished mss.

[Gundel 74] Gundel, Jeanette K. *Role of Topic and Comment in Linguistic Theory.* PhD thesis, University of Texas at Austin, 1974.

[Gundel 80] Gundel, Jeanette K. Zero-NP Anaphora in Russian. *Chicago Linguistic Society Parasession on Pronouns and Anaphora* , 1980.

[Halliday 76] Halliday, Michael A. K. and Hasan, Ruqaiya. *Cohesion in English.* Longman, London, 1976.

[Hawkins 78] Hawkins, John A. *Definiteness and Indefiniteness.* Humanities Press, Atlantic Highlands, New Jersey, 1978.

[Heim 82] Heim, Irene R. *The Semantics of Definite and Indefinite Noun Phrases.* PhD thesis, University of Massachusetts, Amherst, MA, September 1982.

[Hirschman 89] Hirschman, Lynette, Palmer, Martha, Dowding, John, Dahl, Deborah, Linebarger, Marcia, Passonneau, Rebecca, Lang, François-Michel, Ball, Catherine, and Weir, Carl. The PUNDIT Natural-Language Processing System. In *AI Systems in Government Conference*, Computer Society of the IEEE. March 1989.

[Linebarger 88] Linebarger, Marcia C., Dahl, Deborah A., Hirschman, Lynette, and Passonneau, Rebecca J. Sentence Fragments Regular Structures. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics.* Buffalo, NY, June 1988.

[Palmer 85] Palmer, Martha. *Driving Semantics for a Limited Domain.* PhD thesis, University of Edinburgh, 1985.

[Passonneau 88] Passonneau, Rebecca J. A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics* 14(2):44–60, 1988.

[Prince 81] Prince, Ellen F. *Toward a Taxonomy of Given-New Information.* Academic Press, New York, 1981.

[Sidner 79] Sidner, C.L. *Towards a computational theory of definite anaphora comprehension in English discourse.* PhD thesis, MIT, 1979.

[Webber 78] Webber, Bonnie Lynn. *A Formal Approach to Discourse Anaphora.* Garland, New York, 1978.

[Webber 83] Webber, Bonnie Lynn. So What Can We Talk About Now? In Brady, Michael and Berwick, Robert C. (editors), *Computational Models of Discourse*, pages 331–372. MIT Press, 1983.

REDUCING SEARCH BY PARTITIONING THE WORD NETWORK

John Dowding
Unisys Paoli Research Center
P.O. Box 517
Paoli, PA 19355
dowding@prc.unisys.com

This paper proposes an architecture for integrating speech recognition and natural language processing to provide a spoken language understanding system. This work, done in collaboration with the MIT Spoken Language Systems Group, has enabled us to interface our Pundit natural language processing system [Dahl1987,Palmer1986,Hirschman1982,Hirschman1986,Dowding1987] and SUMMIT, the MIT speech recognition system[Zue1989,Glass1988,Seneff1985,Zue1985]. *Information* is passed between the two systems using the *Word Network* (or *Word Lattice*) which is a set of word-score pairs together with the start and end points for each word. The word network is organised as a directed acyclic graph, whose arcs are labeled as word-score pairs, and whose nodes are moments in time. The recognition problem is to find the best scoring grammatical sequence of words from the graph's begin-point to its end. In our experments, we analysed word networks from the TIMIT domain and the Resource Management domain, *constructed* without using any language model.

A spoken language system requires the coupling of speech recognition capabilities with a capability for understanding the meaning of the utterance as provided by a natural language understanding module. The architecture of such a system is an active topic of research. One approach is to simply couple the two components sequentially: the speech recognizer converts the speech signal into a sequence of words, and then the natural language system attempts to understand those words. The criticism of this approach is that natural language system has no opportunity to correct the errors made by the recognizer; nor can it help in controlling the search space. A variant to the this approach is to have the speech recogniser produce a ranked set of candidate sentences. These sentences are then be processed by the natural language system, and the best scoring grammatical (or meaningful) sentence accepted. Alternatively, the two systems can be integrated at a much deeper level, allowing natural language constraints to provide assistance to the recognizer by scoring sentence prefixes as they are produced by the recognizer; this would have the effect of using the grammar to prune the search space, while also producing a representation of the meaning of the utterance. This latter approach -- close interleaving of the speech and natural language components -- is our architectural goal. However, in order to test the effectiveness of these ideas, we have experimented with a sequential coupling of the two systems where the speech recognizer produces a word network, from which candidate sentences are extracted and processed by the natural language component.

Our initial problem in coupling the MIT speech recognition system with the Unisys PUNDIT natural language system was to define an appropriate interface for experimentation. The MIT speech recognizer uses a method called *Viterbi Search* to find the highest scoring sentence. This search strategy is able to find the highest scoring sentence, but it can not be used in its current form to find successively lower scoring sentences. If we had adhered to this strategy, the natural language system might have accepted or rejected the unique output of the speech recognition component, but, as described above, could not have participated in defining a more intelligent search.

In order to allow the natural language system to examine more than one top-scoring sentence candidate, MIT modified their speech recogniser to produce the word network. By developing a strategy to

generate candidate sentences from the word network, it is now possible to couple PUNDIT to this system as a filter. The recogniser generates the word network, a search procedure traverses this network, producing candidate sentences (or sentence prefixes) and PUNDIT accepts or rejects these sentences (or sentence prefixes). Thus if the top scoring string of words turns out not to be a meaningful sentence, there may still be an acoustically lower scoring candidate that is meaningful and receives an analysis from PUNDIT.

Long term, this architecture will permit us to couple PUNDIT to the word network search, to provide its filtering as the word string is built up. These results, however, report on the use of PUNDIT tc filter entire candidate sentences. We have done some experiments on the use of PUNDIT to prune sentence prefixes, and have found that this tends to underestimate the power of natural language constraints, since these constraints are much stronger at the end of the sentence than they are for any of its prefixes.

The main contribution of this paper is to outline the search strategy used to traverse the word network at the interface between the speech recogniser and the natural language component. We experimented with various search strategies to determine the best approach to search the word networks. These strategies included a mix of admissible (breadth-first and best-first) and inadmissible (beam search) strategies. We found that all of the admissible search strategies were too inefficient to be practical. They could not deal with sentence lengths greater than 4 words before they became too slow for our current hardware (Sun 3/60 Workstations). Using an inadmissible search, we could increase efficiency by reducing the beam size, but then we could not get the correct answer. We experienced the phenomenon that the highest scoring path in a beam early in the search would have so many descendents that it would totally dominate the beam later in the search, yielding beams where the alternative candidates were very similar. Because the search was performed strictly left to right, all of the candidates in the beam would share the same left prefix. A similar phenomenon occurs when using an island-driven strategy, except that the common portion may not occur on the left.

We then designed an alternative approach that does not suffer from these problems. This approach maintains a queue of high-scoring candidates that may come from any part of the word network, and that may not be similar to each other. Despite the fact that this approach also uses a strict left to right search, there is no left prefix bias among the high scoring candidates. We parti ion the word network such that all words that have the same start and end points belong to the same partition. Within each partition, the word-score pairs are sorted by score. The score of a partition is defined to be the score of its highest scoring word. This partitioning dramatically reduces the size of the graph that we are searching. For instance, the graph for the sentence "Barb's gold bracelet was a graduation present." contains 1868 arcs. The resulting graph after partitioning contained only 489 arcs (with the same number of nodes in both graphs). This partitioning reduces the "bushiness" of the graph, allowing traditional search procedures to be effective. Figure 1 contains a part of a sample partition from this word network. For 6 of the 8 partitions in this partition path, the correct word (including the pause) was the top candidate within its partition. Notice however that there is still a significant amount of search remaining due to the difference between the scores for "large" and "barb's" in partition 2, especially when compared to the differences between "a", "and", "if", "in", and "an" in partition 6.

Under this scheme, the search for the correct path through the word network is done in two parts: First, the partitioned graph is searched to find the highest scoring partition paths. This search can be done using either an admissible or a beam search strategy. Second, the high scoring sentences are extracted from the set of high scoring partition paths. The sentences are extracted one at a time in order of highest score until one is found that is acceptable to the natural language components.

The search for the highest scoring partitions can be done very quickly, and the beam size can be very small. The worst-case performance of this search is quite good. The amount of time that it takes to find the highest scoring partitions increases with sentence length and beam size, but is nearly unaffected by vocabulary size (worst case growth is O(N log N) as vocabulary increases, but only due to the need to sort the word-score pairs within each partition). Currently, the score of a partition path is the sum of the score of the highest scoring word in each partition. However this algorithm is independent of the particular scoring algorithm used to combine word scores into word-path scores. We plan on

1		2		3		4		5		6		7		8	
-pau-	-9	large	-58	caught	-87	bracelet	-102	was	-51	a	-23	graduation	-157	present	-138
		i'd	-116	gold	-124	geese	-155	with	-89	and	-23	countryside	-189	pairs	-143
		right	-160	could	-134					if	-23			paper	-147
		like	-165							in	-23				
		guard	-193							an	-86				
		barb's	-188												

Figure 1. Sample Partition for "Barb's gold bracelet was a graduation present"

experimenting with more sophisticated scoring techniques, including density and short-fall scoring[Woods1982], in the future.

Extracting the highest scoring sentences from the high scoring partition paths is also done efficiently. The algorithm to do this *is simple:* The partition paths are maintained in a priority queue. To find the highest scoring sentence, the top partition path is removed from the queue and its highest scoring sentence is extracted and reported as the highest scoring sentence in the queue. Then the second highest scoring sentence in the partion path is found, and its score becomes the new score for that partition path. It is then returned to the priority queue based on its new score. The loop is repeated as often as necessary until a sentence is found that is acceptable to the natural language components. While the worst-case performance of this part of the search is not good (the amount of time it takes to find the next highest scoring sentence can grow exponentially), the practical performance is much better, and is able to find the top candidates very quickly. Those cases in which the correct sentence gets a very low score will take a long time to find, but that will be true of any search.

We have tested this interface on word networks from both the Timit and Resource Management domains. We computed the word accuracy for 10 Resource Management word networks chosen randomly from the test set. These networks averaged 4600 arcs (drawn from a vocabulary of 991 words) for sentences ranging in length from 4-11 words. The word accuracy figures for these networks are reported in Figure 2. For comparison, the word accuracy of the SUMMIT system on the same 10 sentences is included. These figures deserve some explanation. The word accuracy figure for the no-

Word Networks		
	Word Accuracy	Perplexity
No Grammar	51%	1126
Syntax	64%	1064
Word-Pair Grammar	85%	60
SUMMIT		
No Grammar	61%	991
Word-Pair Grammar	86%	60

Figure 2. Word Accuracy Results

grammar case is computed by comparing the highest scoring candidate to the correct sentence. We then computed the word accuracy for Pundit by having the search procedure generate candidates one at a time until one was found that was acceptable to Pundit. This experiment used only Pundit's syntactic component. We expect better results using Pundit's semantic and pragmatic components and will report on these results at a subsequent meeting. Finally, we computed the word accuracy using our word network traversal procedure but with the word-pair grammar in place of Pundit's syntactic component.

The difference in word accuracy between the no-grammar case for the word networks (51%) and for SUMMIT (61%) is attributable to two causes. First, the word networks used in our experiments are not complete. They were generated by computing the best score for each word ending at each point. A complete network would have to compute the best score for each word at every beginning and end point. The SUMMIT system does not use an explicit word network, but has access to the complete set of begin and end points for all words. Second, our search for the highest scoring candidate used a beam search (beam size = 500) which is not an admissible search. When computing the word accuracy score for the word-pair grammar, only eight of the ten networks produced candidates that were acceptable to the word-pair grammar within the top 5000 candidates. We expect that this behavior will not occur when the grammar checking and word network search are fully interleaved, because the grammar will prune away ungrammatical paths earlier, permitting well-formed candidates to make it into the beam. For comparison, the performance of Pundit's grammar on the same 8 networks was 72%.

Also in Figure 2 are the perplexity results for Pundit's grammar. This number (1064) appears larger than the vocabulary size (991). We have added vocabulary items representing idiomatic expressions to increase the total vocabulary size to 1126. There are several reasons why this perplexity is so high. The grammar used by Pundit is a very broad coverage grammar of English, including sentence fragments and compound sentences. Also, the lexical items were defined keeping their complete English definition in mind. We are developing a methodology for using the training data for a domain to automatically constrain the broad coverage grammar and lexicon. Finally, our grammar of English provides much stronger constraints at the end of a sentence than it does at any other point. Perplexity only captures constraints that provide immediate pruning of a path. If a constraint would eventually block a path, this is not captured in the perplexity computation. Despite the high perplexity of the Pundit grammar, 77% of the sentence candidates produced by the network traversal procedure were rejected by Pundit.

We are very optimistic that this search procedure will scale up to use in a fully intergrated Spoken Language system. In the prototype, pruning of the search space by the natural language components was only done as the very last step in the search. It is possible to implement this search such that the pruning is done incrementally through the search. This will allow us to both reduce the beam size, and increase the probability that the correct answer will be one of the top candidates at the end of the search.

I would like to acknowledge several people who assisted in this work. Michael Phillips of the MIT Spoken Language Systems Group constructed the word networks, and gave very helpful advice on how they should be searched. Lynette Hirschman, Deborah Dahl, and Shirley Steele read an earlier version of this paper, and gave helpful comments. Francois-Michel Lang helped me find an efficient algorithm for extracting high scoring sentences from a partition path.

References

Dahl1987

Deborah A. Dahl, John Dowding, Lynette Hirschman, Francois Lang, Marcia Linebarger, Martha Palmer, Rebecca Passonneau, and Leslie Riley, Integrating Syntax, Semantics, and Discourse: DARPA Natural Language Understanding Program, R&D Status Report, Paoli Research Center, Unisys Defense Systems, May 14, 1987.

Dowding1987

John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, Presented at the 2nd International Workshop on Natural Language Understanding and Logic Programming, Vancouver, B.C., Canada, 1987.

Glass1988

James R. Glass and Victor W. Zue, Multi-Level Acoustic Segmentation of Continuous Speech, Presented at the International Conference on Acoustics, Speech, and Signal Processing, New York, NY, April 11-14, 1988.

Hirschman1982

L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.

Hirschman1986

L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* **3**(4), 1986, pp. 299-328.

Palmer1986

Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

Seneff1985

Stephanie Seneff, Pitch and Spectral Analysis of Speech Based on an Auditory Synchrony Model, PhD Thesis, Massachusetts Institute of Technology, 1985.

Woods1982

W.A. Woods, Optimal Search Strategies for Speech Understanding Control. *Artificial Intelligence* **18**, 1982.

Zue1989

Victor Zue, James Glass, Michael Phillips, and Stephanie Seneff, Acoustic Segmentation and Phonetic Classification in the SUMMIT System, To Be Presented at the International Conference on Acoustics, Speech, and Signal Processing, Glasgow, Scotland, May 23-26, 1989.

Zue1985

Victor W. Zue, The Use of Speech Knowledge in Automatic Speech Recognition. *Proceedings of the IEEE* **73**(11), 1985, pp. 1602-1615.

Finin, Tim, Richard Fritsson and David Matuszek. "Adding Forward Chaining and Truth Maintenance to Prolog". In *Proceedings of the Fifth Conference on Artificial Intelligence Applications*, March 1989.

Adding Forward Chaining and Truth Maintenance to Prolog

Tim Finin
Rich Fritzson
Dave Matuszek

Logic-Based Systems
Paoli Research Center
Unisys Corporation
Paoli, Pennsylvania

PRC-LBS-8802

September 1988

Abstract

This paper describes P_{fc} , a simple package which supplies a *forward chaining* facility in Prolog. P_{fc} is intended to be used in conjunction with ordinary Prolog programs, allowing the programmer to decide whether to encode a rule as a forward-chaining P_{fc} rule or a backward chaining Prolog one. Like other logic programming languages, P_{fc} programs have a declarative interpretation as well as clear and predictable procedural one. A truth maintenance system is built into P_{fc} system which maintains consistency as well as makes derivations available for applications. Finally, P_{fc} is designed to be practical, being relatively efficient and fairly unobtrusive.

A version of this paper appeared in the the Fifth IEEE Conference on Artificial Intelligence Applications [9].

Please send all correspondance to Tim Finin, Unisys Paoli Research Center, PO Box 517, Paoli, PA 19301; phone: 215-648-7446; email: finin@prc.unisys.com.

1 Introduction

Prolog, like most logic programming languages, offers backward chaining as the only reasoning scheme. It is well known that sound and complete reasoning systems can be built using either exclusive backward chaining or exclusive forward chaining [20]. Thus, this is not a theoretical problem. It is also well understood how to "implement" forward reasoning using an exclusively backward chaining system and vice versa. Thus, this need not be a practical problem. In fact, many of the logic-based languages developed for AI applications [15,7,21,12] allow one to build systems with both forward and backward chaining rules.

There are, however, some interesting and important issues which need to be addresses in order to provide the Prolog programmer with a practical, efficient, and well integrated facility for forward chaining. This paper describes such a facility, P_{fc} , which we have implemented in standard Prolog.

The P_{fc} system is a package that provides a forward reasoning capability to be used together with conventional Prolog programs. The P_{fc} inference rules are Prolog terms which are asserted as facts into the regular Prolog database. For example, Figure 1 shows a file of P_{fc} rules and facts which are appropriate for the ubiquitous kinship domain.

```
spouse(X,Y) <=> spouse(Y,X).
spouse(X,Y),gender(X,G1),{otherGender(G1,G2)}
     =>gender(Y,G2).
gender(P,male) <=> male(P).
gender(P,female) <=> female(P).
parent(X,Y),female(X) <=> mother(X,Y).
parent(X,Y),parent(Y,Z) => grandparent(X,Z).
grandparent(X,Y),male(X) <=> grandfather(X,Y).
grandparent(X,Y),female(X) <=> grandmother(X,Y).
mother(Ma,Kid),parent(Kid,GrandKid)
      =>grandmother(Ma,GrandKid).
grandparent(X,Y),female(X) <=> grandmother(X,Y).
parent(X,Y),male(X) <=> father(X,Y).
mother(Ma,X),mother(Ma,Y),{X\==Y}
     =>sibling(X,Y).
```

Figure 1: **Examples of P_{fc} rules which represent common kinship relations**

In the next section of this paper we will review the nature of forward chaining, the motivations for using it and some of the issues one must address in a forward chaining system. We will the give an informal, example-based presentation of the P_{fc} language in the third section. The fourth section will give some more detailed examples of the use of P_{fc} . Some of the implementation details are described in the fifth section. This paper concludes with a final section which summarizes our experiences and plans for the future.

2 Forward Chaining

It is well known that one can define sound and complete inferencing systems which rely solely on either forward or backward reasoning. This is discussed in introductory AI texts [20]. It is equally well understood that there are good reasons why one might wish to rely on one or the other or a mixed strategy in practice [16,24]. This section reviews some of the reasons why one might want to apply certain knowledge in the form of forward-chaining rules and looks at some of the issues surrounding the design of a forward chaining facility. In particular, we will discuss the importance of a *truth maintenance system* in a forward chaining system.

2.1 Types of Forward Chaining

The terms *forward chaining* and *backward chaining* are used somewhat loosely within the AI community covering such disparate things as OPS5-like production systems to resolution-based theorem provers. In general, when one speaks of a forward chaining rule, one has in mind a rule of the form:

$$P_1 P_2 \ldots P_n \rightarrow Q_1 Q_2 \ldots Q_n$$

Informally, the P_i on the *left hand side* (lhs) of the rule form a set of conditions which, if satisfied, enable the rule for firing. When fired, the Q_i on the rule's *right hand side* (rhs) specify a set of actions which are to be carried out. Typically, the conditions correspond to assertions in a database of facts. A condition is satisfied if it matches some particular fact in this database. There is more variability on the meaning and character of an action in the rhs of a rule. In some systems, such as OPS5 [11,18], these can be arbitrary evaluable expressions. In others, such as MRS, the Q_i are propositions which are to be added to the fact database. Even in this latter case, systems differ as to whether the newly derived facts are persistent and are recorded into the global database [7,15] or are held in a temporary extension to the permanent database (as in a list) [19].

Some of the ways in which the meaning of a forward chaining rule can vary are:

- Are the constituents of the rule's rhs *actions* to be performed or *propositions* to be added to the database?
- When the addition of a new fact to the database triggers some rules, should all of the rules be run or

just one? If more than one is run, should the order be important?
- Should retracting database assertions be allowed?
- If retractions are allowed, how should it affect partially triggered rules?
- If retractions are allowed, should we perform "truth maintenance"?

Given that we are trying to provide a forward chaining facility for Prolog, some of these choices are fairly clear cut. The particulars of the package that we have designed are described in the sections to follow.

2.2 Motivations for Forward Chaining

There are a number of situations in which a forward chaining control strategy is prefered over a backward chaining one. We will briefly mention the major ones. In doing this, we will assume that we are discussing a forward chaining system in which the rhs's of rules specify necessarily true facts which are to be asserted into the global database and remain there until explicitly retracted. The usual reasons for choosing a forward reasoning strategy stem from the particular nature of the problem being solved. These include:

- *space/time tradeoffs.* If a problem involves solving the same goal frequently, it may be more efficient to express the rules for deducing that goal as forward chaining rules. This reduces the computation while increasing the need for static memory.
- *the shape of the inferential space.* Forward chaining tends to involve less search than backward chaining when the ratio of premises to conclusions is high.
- *avoiding long deductive chains.* Forward chaining is useful for avoiding long (or even infinite) deductive chains.
- *drawing all possible inferences.* Many problems require one to draw all possible inferences from a set of axioms. Forward chaining is an efficient way to do this.
- *alerting.* A related situation is one in which we want to ensure that certain deductions are made *as soon as possible*, e.g. for purposes of alerting or monitoring.

There are other reasons to consider using a forward chaining control structure, one of which we will briefly mention. It is sometimes very convenient to construct a logic program in which a subset of the database acts as the interface between two independent and "concurrent" subsystems. As an example, consider designing a system in which one subsystem maintains a model of a set of objects and their locations on a plane and another subsystem is responsible for maintaining a graphical display of these objects. One way to (loosely) couple these two subsystems is to have the first make database assertions which represent the properties of the objects and their locations. The second subsystem has a set of forward chaining rules which are triggered by the first subsystem's assertions and cause the display to be updated. This is similar to the notion of an *active value* found in many Lisp-based systems (e.g. Loops [2], KEE [1]). This was one of the chief ways in which the forward chaining rules of MICRO-PLANNER [23] (the THANTE rules) were used.

2.3 Truth Maintenance

A forward chaining facility in a logic-oriented rule-based system (as opposed to a production-rule oriented system like OPS5) has a special need for a *truth maintenance system* (TMS). Although a TMS can be used with a deductive language using any sort of reasoning strategy, most of the work in TMS systems is associated with forward reasoning [6,5,3,17,14].

There are really two closely related reasons for this need: one *"external"* and the other *"internal"*. The first (the "external") is that we want, in general, to keep the database consistent. Consider a situation in which a forward chaining rule has made an inference based on the state of the database at some time and thereby adds a new fact to the database. If the database changes at some later time such that the earlier inference is no longer valid, then the conclusion may have to be withdrawn and the assertion removed from the database. This is not a problem in a pure backward-chaining system since conclusions are always drawn (and re-drawn, if necessary) with respect to the current state of the database. The second reason (the "internal") has to do with the particular implementational strategy we have employed. Given a rule which has a conjoined lhs, as in:

```
P, Q, R => S
```

one way to encode this is as:

```
P => (Q => (R => S))
```

That is, as a rule with a simple lhs (P) which, when triggered, adds a new rule which monitors for the remaining conjuncts from the original rule. In such a scheme, it is important to keep track of these *derived* rules which represent partially instantiated or triggered rules. If a database assertion supporting a partially triggered rule is erased, then the rule need to be killed. This ensures that a rule will only fire if *all* of its required conditions are simultaneously true.

A truth maintenance system is useful for reasons besides keeping the knowledge base consistent. A TMS

typically records the proof derivations of all of the facts in the database. This is useful in a variety of ways, such as generating explanations, abductive reasoning and debugging.

3 The P_{fc} language

This section describes P_{fc} . We will start by introducing the language informally through a series of examples drawn from the domain of kinship relations. This will be followed by an example and a description of some of the details of its current implementation.

Overview

The P_{fc} package allows one to define forward chaining rules and to add ordinary Prolog assertions into the database in such a way as to trigger any of the P_{fc} rules that are satisfied. An example of a simple P_{fc} rule is:

```
gender(P,male) => male(P)
```

This rule states that whenever the fact unifying with $gender(P, male)$ is added to the database, then the fact $male(P)$ is true. If this fact is not already in the database, it will be added. In any case, a record will be made that the validity of the fact $male(P)$ depends, in part, on the validity of this forward chaining rule and the fact which triggered it. To make the example concrete, if we add $gender(john, male)$, then the fact $male(john)$ will be added to the database unless it was already there.

In order to make this work, it is necessary to use the predicate *add/1* rather than *assert/1* in order to assert P_{fc} rules and any facts which might appear in the lhs of a P_{fc} rule.

Compound Rules

A slightly more complex rule is one in which the rule's left hand side is a conjunction or disjunction of conditions:

```
parent(X,Y),female(X) => mother(X,Y)
mother(X,Y);father(X,Y) => parent(X,Y)
```

The first rule has the effect of adding the assertion $mother(X, Y)$ to the database whenever $parent(X, Y)$ and $female(X)$ are simultaneously true for some X and Y. Again, a record will be kept that indicates that any fact $mother(X, Y)$ added by the application of this rule is justified by the rule and the two triggering facts. If any one of these three clauses is removed from the database, then all facts solely dependent on them will also be removed. Similarly, the second example rule derives the parent relationship whenever either the mother relationship or the father relationship is known.

In fact, the lhs of a P_{fc} rule can be an arbitrary conjunction or disjunction of facts. For example, we might have a rule like:

```
P, (Q;R), S => T
```

P_{fc} handles such a rule by putting it into conjunctive normal form. Thus the rule above is the equivalent to the two rules:

```
P,Q,S => T
P,R,S => T
```

Bi-conditionals

P_{fc} has a limited ability to express bi-conditional rules, such as:

```
mother(P1,P2) <=> parent(P1,P2), female(P1).
```

In particular, adding a rule of the form `P<=>Q` is the equivalent to adding the two rules `P=>Q` and `Q=>P`. The limitations on the use of bi-conditional rules stem from the restrictions that the two derived rules be valid horn clauses. This is discussed in a later section.

Conditioned Rules

It is sometimes necessary to add some further condition on a rule. Consider a definition of sibling which states:

> Two people are siblings if they have the same mother and the same father. No one can be his own sibling.

This definition could be realized by the following P_{fc} rule

```
mother(Ma,P1), mother(Ma,P2), {P1\==P2},
  father(Pa,P1), father(Pa,P2)
   => sibling(P1,P2).
```

Here we must add a condition to the lhs of the rule which states the the variables $P1$ and $P2$ must not unify. This is effected by enclosing an arbitrary Prolog goal in braces. When the goals to the left of such a bracketed condition have been fulfilled, then it will be executed. If it can be satisfied, then the rule will remain active, otherwise it will be terminated.

Negation

We sometimes want to draw an inference from the absence of some knowledge. For example, we might wish to encode the default rule that a person is assumed to be male unless we have evidence to the contrary:

```
person(P), ~female(P) => male(P).
```

A lhs term preceded by a $\sim$ is satisfied only if *no* fact in the database unifies with it. Again, the P_{fc} system records a justification for the conclusion which, in this case, states that it depends on the absence of the contradictory evidence. The behavior of this rule is demonstrated in the following dialogue:

```
?- add(person(P), ~female(P) => male(P)).
yes
?- add(person(alex)).
yes
?- male(alex).
yes
?- add(female(alex)).
yes
?- male(alex)
no
```

As a slightly more complicated example, consider a rule which states that we should assume that the parents of a person are married unless we know otherwise. Knowing otherwise might consist of either knowing that one of them is married to a yet another person or knowing that they are divorced. We might try to encode this as follows:

```
parent(P1,X),
parent(P2,X),
{P1\==P2},
~divorced(P1,P2),
~spouse(P1,P3),
{P3\==P2},
~spouse(P2,P4),
{P4\==P1}
  =>
spouse(P1,P2).
```

Unfortunately, this won't work. The problem is that the conjoined condition

```
~spouse(P1,P3),{P3\==P2}
```

does not mean what we want it to mean - that there is no $P3$ distinct from $P2$ that is the spouse of $P1$. Instead, it means that $P1$ is not married to any $P3$. We need a way to move the qualification {P3\==P2} inside the scope of the negation. To achieve this, we introduce the notion of a qualified goal. A lhs term P/C, where P is a positive atomic condition, is true only if there is a database fact unifying with P and condition C is satisfiable. Similarly, a lhs term $\sim P/C$, where P is a positive atomic condition, is true only if there is no database fact unifying with P for which condition C is satisfiable. Our rule can now be expressed as follows:

```
parent(P1,X),
  parent(P2,X)/(P1\==P2),
  ~divorced(P1,P2),
  ~spouse(P1,P3)/(P3\==P2),
  ~spouse(P2,P4)/(P4\==P1)
  =>
  spouse(P1,P2).
```

Procedural Interpretation

Note that the procedural interpretation of a P_{fc} rule is that the conditions in the lhs are checked *from left to right*. One advantage to this is that the programmer can chose an order to the conditions in a rule to minimize the number of partial instantiations. Another advantage is that it allows us to write rules like the following:

```
at(Obj,Loc1),at(Obj,Loc2)/{Loc1\==Loc2}
   => {remove(at(Obj,Loc1))}.
```

Although the declarative reading of this rule can be questioned, its procedural interpretation is clear and useful:

> If an object is known to be at location *Loc*1 and an assertion is added that it is at some location *Loc*2, distinct from *Loc*1, then the assertion that it is at *Loc*1 should be removed.

The Right Hand Side

The examples seen so far have shown a rules rhs as a single proposition to be "added" to the database. The rhs of a P_{fc} rule has some richness as well. The rhs of a rule is a conjunction of facts to be "added" to the database and terms enclosed in brackets which represent conditions/actions which are executed. As a simple example, consider the conclusions we might draw upon learning that one person is the mother of another:

```
mother(X,Y) =>
  female(X),
  parent(X,Y),
  adult(X).
```

As another example, consider a rule which detects bigamists and sends an appropriate warning to the proper authorities:

```
spouse(X,Y), spouse(X,Z), {Y\==Z} =>
   bigamist(X),
   {format("~N~w is a bigamist, married
      to both ~w and ~w~n",[X,Y,Z])}.
```

Each element in the rhs of a rule is processed from left to right — assertions being added to the database with appropriate support and conditions being satisfied. If a condition can not be satisfied, the rest of the rhs is not processed.

We would like to allow rules to be expressed as bi-conditional in so far a possible. Thus, an element in

the lhs of a rule should have an appropriate meaning on the rhs as well. What meaning should be assigned to the conditional fact construction (e.g. P/Q) which can occur in a rules lhs? Such a term in the rhs of a rule is interpreted as a *conditioned assertion*. Thus the assertion P/Q will match a condition $P\prime$ in the lhs of a rule only if P and $P\prime$ unify and the condition Q is satisfiable. For example, consider the rules that says that an object being located at one place is reason to believe that it is not at any other place:

```
at(X,L1) => not(at(X,L2))/L2\==L1
```

Note that a *conditioned assertion* is essentially a Horn clause. We would express this fact in Prolog as the backward chaining rule:

```
not(at(X,L2)) :- at(X,L1),L1\==L2.
```

The difference is, of course, that the addition of such a conditioned assertion will trigger forward chaining whereas the assertion of a new backward chaining rule will not.

The Truth Maintenance System

As discussed in the previous section, a forward reasoning system has special needs for some kind of *truth maintenance system*. The P_{fc} system has a rather straightforward TMS system which records justifications for each fact deduced by a P_{fc} rule. Whenever a fact is removed from the database, any justifications in which it plays a part are also removed. The facts that are justified by a removed justification are checked to see if they are still supported by some other justifications. If they are not, then those facts are also removed.

Such a TMS system can be relatively expensive to use and is not needed for many applications. Consequently, its use and nature are optional in P_{fc} and are controlled by the predicate $pfcTmsMode/1$. The possible cases are three:

- $pfcTmsMode(full)$ - The fact is removed unless it has *well founded support* (WFS). A fact has WFS if it is supported by the *user* or by *God* or by a justification all of whose justificees have WFS[1].
- $pfcTmsMode(local)$ - The fact is removed if it has no supporting justifications.
- $pfcTmsMode(none)$ - The fact is never removed.

A fact is considered to be supported by *God* if it is found in the database with no visible means of support. That is, if P_{fc} discovers an assertion in the database that can take part in a forward reasoning step, and that assertion is not supported by either the user or a forward deduction, then a note is added that the assertion is supported by *God*. This adds additional flexibility in interfacing systems employing P_{fc} to other Prolog applications.

For some applications, it is useful to be able to justify actions performed in the rhs of a rule. To allow this, P_{fc} supports the idea of declaring certain actions to be *undoable* and provides the user with a way of specifying methods to undo those actions. Whenever an action is executed in the rhs of a rule and that action is undoable, then a record is made of the justification for that action. If that justification is later invalidated (e.g. through the retraction of one of its justificees) then the support is checked for the action in the same way as it would be for an assertion. If the action does not have support, then P_{fc} trys each of the methods it knows to undo the action until one of them succeeds.

In fact, in P_{fc} , one declares an action as undoable just by defining a method to accomplish the undoing. This is done via the predicate $pfcUndo/2$. The predicate $pfcUndo(A1, A2)$ is true if executing $A2$ is a possible way to undo the execution of $A1$. For example, we might want to couple an assertional representation of a set of graph nodes with a graphical display of them through the use of P_{fc} rules:

```
at(N,XY) => {displayNode(N,XY)}.
arc(N1,N2) => {displayArc(N1,N2}.

pfcUndo(displayNode(N,XY),eraseNode(N,XY)).
pfcUndo(displayArc(N1,N2),eraseArc(N1,N2)).
```

Limitations

The P_{fc} system has several limitations, most of which it inherits from its Prolog roots. One of the more obvious of these is that P_{fc} rules must be expressible as a set of horn clauses. The practical effect is that the rhs of a rule must be a conjunction of terms which are either assertions to be added to the database or actions to be executed. Negated assertions and disjunctions are not permitted, making rules like

```
parent(X,Y) <=> mother(X,Y);father(X,Y)
male(X) <=> ~female(X)
```

ill-formed.

Another restrictions is that all variables in a P_{fc} rule have implicit universal quantification. As a result, any variables in the rhs of a rule which remain uninstantiated when the lhs has been fully satisfied retain their universal quantification. This prevents us from using a rule like

[1] Determining if a fact has WFS requires detecting local cycles - see [16] for an introduction

```
father(X,Y), parent(Y,Z)
    <=> grandfather(X,Z).
```

with the desired results. If we do add this rule and assert *grandfather(john,mary)*, then P_{fc} will add the two independent assertions *father(john,_)* (i.e. "John is the father of everyone") and *parent(_,mary)* (i.e. "Everyone is Mary's parent").

Another problem associated with the use of the Prolog database is that assertions containing variables actually contain "copies" of the variables. Thus, when the conjunction

```
add(father(adam,X)), X=able
```

is evaluated, the assertion `father(adam,_G032)` is added to the database, where _G032 is a new variable which is distinct from X. As a consequence, it is never unified with *able*.

4 Examples

This section gives two examples. The first uses the simple kinship domain to show the network of TMS assertions generated from a small set of assertions. The second shows how P_{fc} can be applied to a typical diagnosis problem.

Figure 2 shows a simple P_{fc} program and Figure 3 shows the resulting network of assertions. In this last figure we have abbreviated the names of the predicates and constants to make the depiction more compact.

```
father(X,Y), parent(Y,Z)
     => grandfather(X,Z).
parent(X,Y), male(X) <=> father(X,Y).
parent(X,Y) <=> child(Y,X).
=> male(tom).
=> parent(tom,tim).
=> child(clare,tim).
=> father(tim,peter).
```

Figure 2: A simple P_{fc} program in the kinship domain

Our second example shows how P_{fc} can be used to provide the representation and reasoning core of a simple diagnostic application. Figure 4 shows a standard diagnostic problem that has been used widely in the literature [8,13,4,22]. The problem is to determine which components are faulty, given the description of the structure of the circuit and the observed input and output values (values shown in parentheses are

abbreviations:
predicates: p=parent, f=father, m=male, gf=grandfather, c=child, s=fcSupports

constants: u=user, p1=tom, p2=tim, p3=clare, p4=peter

user asserted facts:

```
f(X,Y), p(Y,Z) => gf(X,Z).
p(X,Y), m(X) <=> f(X,Y).
p(X,Y) <=> c(Y,X).
m(p2).
p(p2,p2).
c(p3,p2).
f(p2,p4).
```

deduced facts:

```
f(p1,p2).
c(p2,p1).
p(p2,p3).
p(p2,p4).
```

triggers:

```
pt(f(A,B),pt(p(B,C),rhs([gf(A,C)]))).
pt(p(A,B),pt(m(A),rhs([f(A,B)]))).
pt(f(A,B),rhs([p(A,B),m(A)])).
pt(p(A,B),rhs([c(B,A)])).
pt(c(A,B),rhs([p(B,A)])).
pt(m(p1),rhs([f(p1,p2)])).
pt(p(p2,A),rhs([gf(p1,A)])).
pt(m(p2),rhs([f(p2,p3)])).
pt(p(p4,A),rhs([gf(p2,A)])).
pt(m(p2),rhs([f(p2,p4)])).
pt(p(p3,A),rhs([gf(p2,A)])).
```

tms relations:

```
s(nu,(f(A,B),p(B,C)=>gf(A,C))).
s(((f(A,B),p(B,C)=>gf(A,C)),u),pt(f(A,B),pt(p(B,C),rhs([gf(A,C)])))).
s(nu,(p(A,B),m(A)<=>f(A,B))).
s(((p(A,B),m(A)<=>f(A,B)),u),pt(p(A,B),pt(m(A),rhs([f(A,B)])))).
s(((p(A,B),m(A)<=>f(A,B)),u),pt(f(A,B),rhs([p(A,B),m(A)]))).
s(nu,(p(A,B)<=>c(B,A))).
s(((p(A,B)<=>c(B,A)),u),pt(p(A,B),rhs([c(B,A)]))).
s(((p(A,B)<=>c(B,A)),u),pt(c(B,A),rhs([p(A,B)]))).
s(nu,m(p1)).
s(nu,p(p1,p2)).
s((p(p1,p2),pt(p(p1,p2),pt(m(p1),rhs([f(p1,p2)])))),
pt(m(p1),rhs([f(p1,p2)]))).
s((m(p1),pt(m(p1),rhs([f(p1,p2)]))),f(p1,p2)).
s((f(p1,p2),pt(f(p1,p2),pt(p(p2,A),rhs([gf(p1,A)])))),
pt(p(p2,A),rhs([gf(p1,A)]))).
s((f(p1,p2),pt(f(p1,p2),rhs([p(p1,p2),m(p1)]))),p(p1,p2)).
s((f(p1,p2),pt(f(p1,p2),rhs([p(p1,p2),m(p1)]))),m(p1)).
s((p(p1,p2),pt(p(p1,p2),rhs([c(p2,p1)]))),c(p2,p1)).
s((c(p2,p1),pt(c(p2,p1),rhs([p(p1,p2)]))),p(p1,p2)).
s(nu,c(p3,p2)).
s((c(p3,p2),pt(c(p3,p2),rhs([p(p2,p3)]))),p(p2,p3)).
s((p(p2,p3),pt(p(p2,p3),pt(m(p2),rhs([f(p2,p3)])))),
pt(m(p2),rhs([f(p2,p3)]))).
s((p(p2,p3),pt(p(p2,p3),rhs([c(p3,p2)]))),c(p3,p2)).
s((p(p2,p3),pt(p(p2,p3),rhs([gf(p1,p3)]))),gf(p1,p3)).
s(nu,f(p2,p4)).
s((f(p2,p4),pt(f(p2,p4),pt(p(p4,A),
rhs([gf(p2,A)])))),pt(p(p4,A),rhs([gf(p2,A)]))).
s(((f(p2,p4),pt(f(p2,p4),rhs([p(p2,p4),m(p2)]))),p(p2,p4)).
s((p(p2,p4),
pt(p(p2,p4),pt(m(p2),rhs([f(p2,p4)])))),pt(m(p2),rhs([f(p2,p4)]))).
s((p(p2,p4),pt(p(p2,p4),rhs([c(p4,p2)]))),c(p4,p2)).
s((c(p4,p2),pt(c(p4,p2),rhs([p(p2,p4)]))),p(p2,p4)).
s((p(p2,p4),pt(p(p2,p4),rhs([gf(p1,p4)]))),gf(p1,p4)).
s(((f(p2,p4),pt(f(p2,p4),rhs([p(p2,p4),m(p2)]))),m(p2)).
s((m(p2),pt(m(p2),rhs([f(p2,p3)]))),f(p2,p3)).
s((f(p2,p3),
pt(f(p2,p3),pt(p(p3,A),rhs([gf(p2,A)])))),pt(p(p3,A),rhs([gf(p2,A)]))).
s((f(p2,p3),pt(f(p2,p3),rhs([p(p2,p3),m(p2)]))),p(p2,p3)).
s(((f(p2,p3),pt(f(p2,p3),rhs([p(p2,p3),m(p2)]))),m(p2)).
s((m(p2),pt(m(p2),rhs([f(p2,p4)]))),f(p2,p4)).
```

Figure 3: The resulting network of assertions produced by the simple kinship program

Figure 4: A simple circuit to be diagnosed

predictions). The general approach to this kind of problem is to represent the circuit (i.e. the devices, their behavior, and the connections) in the form of logical assertions and/or rules. The representation indicates (implicitly or explicitly) that some of the facts about the circuit are to be taken as assumptions (e.g. that a device behaves as intended or that an observation is correct).

If all of the information about the circuit (structure, device behavior and observations) is consistent, then there is no evidence of a problem and no diagnosis is needed. If there is an inconsistency, however, then one or more of the assumptions must be incorrect. The diagnostic problem, then, can be expressed in terms of finding "minimal" sets of assumptions to retract such that the resulting set of known facts and assumptions is consistent (see [10] for an overview of this approach).

Although P_{fc} was not specifically designed to solve diagnostic problems, it turns out to provide some of the necessary representation and reasoning capabilities. The P_{fc} system supplies a good way to describe a set of facts and to draw all possible conclusions from them (the deductive closure) and to record the derivations of these facts. The first capability is just what is needed to *simulate* the behavior of the circuit to be diagnosed and to identify the *conflicts* that arise. The second capability allows the derivations of these conflicts to be explored, looking for a set of assumptions to be retracted.

The predicates to represent such circuits are given in Figure 5. In this scheme, a device is represented by a symbol. To indicate that a device, d, is a kind of *adder*, we can add the assertion *isa(d,adder)*. The first P_{fc} rule in Figure 4 will then conclude that d exhibits the behavior of an adder (i.e. adding the assertion *behave(d,adder)*) unless it is known that d is defective (i.e. unless there is a fact matching *faulty(d)*. When the fact that d behaves like an adder is asserted, the fifth rule in this figure will add the constraints (expressed as P_{fc} rules) which relate d's inputs and output.

```
% Devices behave as they should unless they are faulty.
isa(X,Class), ~faulty(X) => behave(X,Class).

% a wire equates the values at its two ends.
wire(T1,T2) => (val(T1,V) <=> val(T2,V)).

% It is a conflict if a terminal has two different values.
val(T,V1), val(T,V2)/{\+V1=V2} => conflict(T).

% assume an observation is true.
observed(P), ~false_observation(P) => P.

% an adder's behaviour
behave(X,adder) =>
 (val(in1(X),I1), val(in2(X),I2)
        => {O is I1+I2}, val(out(X),O)),
 (val(in1(X),I1), val(out(X),O)
        => {I2 is O-I1}, val(in2(X),I2)),
 (val(in2(X),I2), val(out(X),O)
        => {I1 is O-I2}, val(in1(X),I1)).

% a multiplier's behaviour.
behave(X,multiplier) =>
 (val(in1(X),I1), val(in2(X),I2)
        => {O is I1*I2}, val(out(X),O)),
 (val(in1(X).I1), val(out(X),O)
        => {I2 is O/I1},val(in2(X),I2)),
 (val(in2(X),I2), val(out(X),O)
        => {I1 is O/I2}, val(in1(X),I1)).

% a gizmo is the standard example circuit.
isa(X,gizmo) =>
  isa(m1(X),multiplier),
  isa(m2(X),multiplier),
  isa(m3(X),multiplier),
  isa(a1(X),adder),
  isa(a2(X),adder),
  wire(out(m1(X)),in1(a1(X))),
  wire(out(m2(X)),in2(a1(X))),
  wire(out(m2(X)),in1(a2(X))),
  wire(out(m3(X)),in2(a2(X))).
```

Figure 5: P_{fc} rules which simulate the behavior of simple circuits composed of adders and multipliers

Such a simulation can be used by the simple program shown below:

```
% test(+case_id,-diagnosis)
test(X,L) :-
 (=> isa(X,gizmo),
  => val(in1(m1(X)),3),
  => val(in2(m1(X)),2),
  => val(in1(m2(X)),3),
  => val(in2(m2(X)),2),
  => val(in1(m3(X)),2),
  => val(in2(m3(X)),2),
  => observed(val(out(a1(X)),6)),
  => observed(val(out(a2(X)),6))),
  diagnosis(X,L).
```

which takes a symbol identifying a case and returns a list of possible hypotheses.

5 Implementation

This section briefly describes the current implementation of P_{fc} . The basic user predicates are *add/1* and *rem/1*. The *add/1* predicate adds a new P_{fc} fact or rule to the database, triggering any forward chaining. Adding a new rule involves putting the rule's lhs into a modified conjunctive normal form and then adding one or more *triggers* to the database. Conceptually, a trigger represents a demon which monitors the database, watching for the addition or removal of an assertion which can unify with its *head*. A trigger also has a *condition* which, if satisfiable, will enable the "evaluation" of its *body*. For example, the rule

```
father(X,Y),parent(Y,Z)=>grandfather(X,Z).
```

results in the following trigger being asserted into the database:

```
pt(father(A,B),
   true,
   pt(parent(B,C),
      true,
      rhs([grandfather(A,C)]))).
```

Whenever a fact is added to the database (for the first time) all positive triggers with unifying heads are collected and fired. Firing a trigger means ensuring that its condition is satisfied and processing the body. The body can be another trigger, a conditional body, a "cut point", or the rule's rhs.

When the body of a trigger is another trigger, it is asserted into the database with a note that it's support comes from the initial trigger and the unifying fact. Thus, in the above example, when *father(tom,tim)* is asserted, the trigger

```
pt(parent(tim,C),
   true,
   rhs([grandfather(tom,C)]))
```

is added to the database with support coming from the original trigger and fact.

An item in the lhs of a rule can be an arbitrary condition wrapped in braces, as in:

```
age(P1,A1),age(P2,A2),{A1>A2}
    => older(P1,P2).
```

This provides additional flexibility in mixing forward and backward reasoning and also makes the semantics of bi-conditional rules sensible.

We are experimenting with a technique for pruning the tree of triggers which grows from a rule and a stream of facts which is being added to the database. This is analogous to the use of the *cut* operation in Prolog and other logic programming languages. For example, consider a rule which encodes the knowledge that a person is a parent if they have offspring. We could write this in P_{fc} as:

```
person(P),parent(P,_) => isParent(P).
```

However, this rule is somewhat redundant in that it records multiple justifications for the *isParent* conclusion. That is if a person has six children, then there will be six justifications for the conclusion. In many applications, it is desirable to "prune" away the other justifications, an operation similar to the "cut" in logic programming languages. In P_{fc} the "!" symbol represents such a pruning operation. We can write our rule as:

```
person(P),parent(P,_),! => isParent(P).
```

Whenever the "!" is encountered in a rule instance, all ancestor triggers "frozen". This effectively blocks any justifications beyond the first. If the first justification is removed by the tms system, the effective triggers will be "thawed".

Finally, the trigger which represents the last condition in a rule will have the rule's rhs as its body. Similarly, whenever a positive trigger is added to the database, it is "fired" for each extant fact in the database with which it unifies. Consider the following rule which contains a negated fact in the lhs:

```
parent(P1,K),spouse(P1,P2),~parent(P2,K)
  => stepParent(P2,K).
```

This rule would generate the following trigger:

```
pt(parent,
   true,
   pt(spouse(P1,P2),
      true,
```

```
nt(parent(P2,K),
   true,
   rhs[stepParent(P2,K)]))
```

The *nt/3* term represents a *negative trigger* which is immediately satisfied if there is no unifying fact in the database. Whenever a fact is removed from the database, all negative triggers with unifying heads are gathered and, if their conditions are satisfiable, fired. Conversely, whenever a fact is added to the database, a search is made for justifications which include a negative trigger whose head unifies with the newly added fact. Any such justifications are then removed.

The support for conclusions is recorded by the *fcSupport/2* predicate. It has one of the following forms:

- *fcSupport((user,user),X)* - where X is a user asserted rule or fact.
- *fcSupport((Rule,user),Trigger)* - where Rule is user-asserted rule and Trigger is one of the resulting initial triggers.
- *fcSupport((Fact,Trigger),X)* - where Fact is an atomic fact, Trigger is a positive or negative trigger and X is a resulting fact or another trigger.

These assertions are hidden from the user in a shadow database.

Other predicates exist for finding the immediate facts and rules which support a given clause and for finding the set of "user asserted" facts and rules which support a clause. These can be used to construct the possible P_{fc} derivations of a clause.

6 Conclusions

This paper has described P_{fc} a forward chaining facility for Prolog. P_{fc} is intended to be used in conjunction with ordinary Prolog programs, allowing the programmer to decide whether to encode a rule as a forward-chaining P_{fc} rule or a backward chaining Prolog one. Like other logic programming languages, P_{fc} programs have a declarative interpretation as well as clear and predictable procedural one. A truth maintenance system is built into P_{fc} system which maintains consistency as well as makes derivations available for applications. Finally, P_{fc} is designed to be practical, being relatively efficient and fairly unobtrusive.

We have begun to experiment with P_{fc} are expecting to use it in several Prolog-based applications requiring a forward reasoning facility. There are a number of issues which we intend to examine in the near future. These include:

- *applications.* We are looking at several applications which have a need for a Prolog-based forward-reasoning capability.
- *controlling forward reasoning.* We have implemented a simple scheme which provides an operation similar to the "cut" operation in Prolog and other logic programming languages. If the cut operator is found in a rule, it propagates a deactivation signal backward through the tree of triggers associated with the rule. This prevents the rule from generating any additional deductions. We would like to study several alternative schemes for controlling forward reasoning and see which appear to be most useful.
- *optimizations and compilation.* There are some fairly straightforward optimizations can can be employed. For example, triggers which have no uninstantiated variables can only be satisfied once. Once satisfied, they can be "frozen", reducing the number of active triggers. We have sketched a technique for transforming parts of P_{fc} programs into regular Prolog rules. This technique would allow these to be compiled. We plan on implementing this scheme and studying the result to see if there is a significant gain in efficiency.
- *parallel execution.* A pure version of a forward chaining system like P_{fc} could be run in parallel. A pure version would eliminate the *Prolog-level* side effects and the cut operation but retain the basic model that P_{fc} rules draw conclusions which are recorded by making assertions into the prolog database. A pure version has the property that the order in which the rules are triggered or assertions made does not matter. We plan on investigating the parallel execution of a pure version of P_{fc} .

In summary, we have found that the P_{fc} system effectively extends Prolog to enable one to use a mixed backward and forward reasoning strategy. This is done in a way that maintains the advantages of using Prolog (as opposed to a more general logic-based AI language) — simplicity, speed and portability.

References

[1] *KEE Reference Manual.* Intellicorp, level 3.0 edition, 1986.

[2] D. G. Bobrow and M. Stefik. *The Loops Manual.* Technical report KB-VLSI-81-13, Xerox PARC, 1981.

[3] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[4] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97 – 130, 1987.

[5] Jon Doyle. The ins and outs of reason maintenance. In 8^{th} *International Conference on Artificial Intelligence*, pages 349–351, 1983.

[6] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[7] M. Genesereth et. al. *MRS Manual.* Technical Report, Stanford University, 1983.

[8] Randall Davis et. al. Diagnosis based on description of structure and function. In *Proc. National Conf. on Artificial Intelligence*, AAAI, CMU, Pittsburgh PA, August 1982.

[9] Tim Finin, Rich Fritzson, and Dave Matuzsek. Adding forward chaining and truth maintenance to prolog. In *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*, March 1989.

[10] Tim Finin and Gary Morris. Abductive reasoning in multiple fault diagnosis. *Artificial Intelligence Review*, 3(2), 1989.

[11] C. L. Forgy. *The OPS5 User's manual.* Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, 1981.

[12] Rich Fritzson and Tim Finin. *Protem - An Integrated Expert Systems Tool.* Technical Report 84, Logic Based Systems, Paoli Research Center, Unisys, 1988.

[13] M. R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[14] David A. McAllester. *Reasoning Utility Package User's Manual.* Technical Report, MIT Artificial Intelligence Laboratory, April 1982.

[15] Drew McDermott. *DUCK: A Lisp-Based Deductive System.* Technical Report, Computer Science, Yale University, 1983.

[16] Drew McDermott and Eugene Charniak. *Introduction to Artificial Intelligence.* Addison Wesley, 1985.

[17] Drew McDermott and Jon Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13(1):41–72, 1980.

[18] Dennis Merritt. Forward chaining in Prolog. *AI Expert*, November 1986.

[19] Paul Morris. A forward chaining problem solver. *Logic Programming Newsletter*, Autumn 1981.

[20] Nils Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, California, 1980.

[21] Charles J. Petrie and Michael N. Huhns. *Controlling Forward Rule Inferences.* Technical Report ACA-AI-012-88, MCC, January 1988.

[22] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 96, 1987.

[23] Gerald J. Sussman, Terry Winograd, and Eugene Charniak. *Micro-planner Reference Manual.* Technical Report AIM 203a, MIT Artificial Intelligence Laboratory, 1971.

[24] Richard Treitel and Michael Genesereth. Choosing directions for rules. *Journal of Automated Reasoning*, 3(4):395–432, Dec. 1987.

Hirschman, L. "A Meta-Treatment of wh-Constructions". *META88: Proceedings of the Workshop on Meta-Programming in Logic Programming*, June, 1988.

Chapter 1
A Meta-Rule Treatment for English Wh-Constructions

Lynette Hirschman[1]

Paoli Research Center
Unisys Defense Systems

Abstract

This paper describes a general meta-rule treatment of English wh-constructions (relative clauses, and questions) in the context of a broad-coverage logic grammar that also includes an extensive meta-rule treatment of co-ordinate conjunction. Wh-constructions pose difficulties for parsing, due to their introduction of a dependency between the wh-word (e.g., *which*) and a corresponding *gap* in the following clause: *This is the book which I thought you told me to refer to ( )*. The gap can be arbitrarily far away from the wh-word, but it *must* occur within the clause, or the sentence is not well-formed, as in **The book which I read it*.

A meta-rule treatment has several advantages over an Extraposition Grammar-style treatment: a natural delimitation of the gap scope, the ability to translate/compile the grammar rules, and ease of integration with conjunction. Wh-constructions are handled by annotating those grammar rules that license a gap or realize a gap. These annotations are converted, via the meta-rule component, into parameterized rules. A set of paired input/output parameters pass the need for a gap from parent to child and left sibling to right sibling until the gap is realized; once the gap is realized, the parameter takes on a *no_gap* value, preventing further gaps from being realized. This 'change of state' in the paired parameters ensures that each gap is filled exactly once. The conjunction meta-rule operates on the parameterized wh-rules to link gaps within conjoined structures by unification, so that any gap within a conjoined structure is treated identically for all conjuncts.

[1]This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research; and in part by internal Unisys funding.

1.1 Introduction

Wh-constructions are one of the classically difficult parsing problems, because a correct treatment requires interaction of non-adjacent constituents, namely the wh-word, which introduces a constituent in clause-initial position, and the following construction which is missing a constituent (the *gap*). The gap can be arbitrarily far from the introducing wh-word (an *unbounded dependency*); in particular, it can appear within deeply embedded constructions, such as *the person that [I had hoped [Jane would tell [( ) to get the books]]]*, where there are three levels of embedded structure. It is possible, in principle, to write a rule for each case where a gap can appear. However, since the number of constructions which can accommodate a gap is very large (e.g., most complement types), this is both extremely labor-intensive and unmaintainable from the grammar writer's point of view.

It is also possible to write general rules for gap-realization, e.g., a noun phrase can be realized as a gap. If this approach is taken, then these rules must be carefully constrained to accept gaps only when inside a wh-construction; in addition, the wh-construction must contain exactly *one* gap. These restrictions involve complex and expensive search up and down the parse tree, to determine whether a gap is occurring inside a wh-construction.

In many ways, the wh-problem parallels the problem of co-ordinate conjunction that has also been a major obstacle for natural language systems. Both constructions involve gaps, both affect large portions of the grammar, and both require a major modification to the grammar and/or to the parsing mechanism to handle the linguistic phenomena.

There have been two basic approaches to conjunction and wh-constructions in the computational linguistics literature: modification of the parser (interpreter) and meta-rules. Of these, the first approach has been far more common. For conjunction, a number of variants on the 'interrupt' driven approach have been presented, both in conventional natural language processing systems [13,12,14], and in the context of logic grammars [4]. The same is true for logic grammar implementations of wh-constructions: the most generally used treatment is the interpreter-based treatment of Extraposition Grammar (XG) [10].

Meta-rules offer an appealing alternative to interpreter-based approaches, both for conjunction and for wh-expressions. Meta-rules are particularly well-suited to phenomena that range over a variety of syntactic structures, where the linguistic description would otherwise require regular changes to a large set of grammar rules. The use of meta-rules turns out to be efficient computationally. It also preserves compactness of the underlying grammar, so that the grammar is still maintainable from the point of view of the grammar-writer. Finally, the meta-rule approach avoids additional interpretive overhead and permits translation/compilation of grammar rules for efficient execution [5].

For conjunction, the meta-rule approach forms the basis for a comprehensive

treatment of co-ordinate conjunction in Restriction Grammar [7]. Abramson has provided a generalization of this approach, formulating meta-rules as a specialized case of meta-programming [2]. Other researchers have also examined a meta-rule approach to related phenomena; Banks and Rayner, for example, have proposed a meta-treatment of the comparative [3].

For wh-constructions, we propose here an approach based on parameterization of the grammar rules. This is similar in spirit to the GPSG notion of 'slash categories' [6], but in the framework of logic grammar. The use of parameterized rules to pass gap information has previously been proposed in a logic grammar framework, specifically as *gap-threading* [10,11]. Our approach differs from Pereira's in several ways, the most important of which is the use of meta-rules. The meta-rule approach provides a much cleaner user interface, making it possible for the grammar writer to use linguistically motivated annotations to indicate gap license and gap realization for the unparameterized BNF definitions in the grammar. The meta-rules process these annotations to generate parameterized grammar rules which, in turn, can be translated and compiled for efficient execution. The meta-rule treatment also has the property of combining seamlessly with a meta-rule treatment of co-ordinate conjunction.

1.2 Wh-Constructions: The Linguistic Issues

Wh-constructions are one instance of a class of problems referred to as *unbounded dependencies* – that is, constructions where the interdependent entities may be arbitrarily far apart. In the case of wh-constructions, we have a wh-expression which begins the clause (e.g., *who, what, which, whose book, how*, etc.) followed by a *gap* at some later point in the clause. The wh-expression may take the place of a noun phrase, an adjective phrase or an adverbial phrase. These may appear in the subject, object or sentence adjunct positions.

As the sentences of Figure 1 illustrate, there are a variety of wh-constructions, namely, relative clauses (including the zero-complementizer case, where an overt wh-word is absent, as in *the person I saw*), indirect questions (*I don't know what they mean*), wh-questions (*What do you want?*), and headless relatives (*You get what you deserve*). In addition to these basic types of wh-construction, there are also some constructions where the wh-expression is embedded inside a noun phrase (*this is the person whose mother I met*), with the wh-word *whose* modifying a noun phrase; the subsequent gap is filled by the noun phrase (*the person's mother*) of which the wh-word is a part. There are also wh-constructions embedded in prepositional phrases, as in *the person from whom I learned it* or *the door the key to which is missing.*

A wh-construction involves (1) a *wh-word* (e.g., *who*) contained in a clause-initial *wh-expression*; and (2) a *gap*: a constituent omitted in the clause following the wh-word, e.g., *the book which I bought ( )*. Relative clauses also have an antecedent for the relative pronoun (the wh-word); for questions, the wh-word

marks the questioned item.

Wh = who; gap = subject NP
The person who () was here
Wh = who(m); gap = object NP
Who did you see ()?
Wh = that; gap = object NP
The time that I spent ()
Wh = that; gap = sentence adjunct adverbial
The time I visited them ()
Wh = who(m); gap = embedded object
The person who they told me they had tried to visit ()
Wh = who; gap = embedded subject
Who did they tell you () had visited them?
Wh = how; gap = sentence adjunct adverbial
Do you know how they did it ()?

Figure 1: Wh-constructions in English

To regularize a wh-construction, the wh-expression fills in the gap, and the wh-word is replaced by its antecedent (if in a relative clause).[1] For example, in the phrase *the movie which I saw ( )*, the wh-expression is *which* and the gap is after *saw*. Moving the wh-expression into the gap, we get *the movie [I saw which]*. Then, replacing *which* by its antecedent (*the movie*), we get: *the movie [I saw the movie]*. Similarly for questions, we get *what did you see ( )?* regularized as *did you see what?*. In some cases, however, the wh-word is not identical to the whole wh-expression, as in *the bird whose nest I found ( )*. Here, the wh-expression is *whose nest*, and the wh-word is *whose*. Again, we replace the gap (the object of *found*) by the wh-expression, to get *the bird [I found whose nest]*. Then we replace the wh-word by its antecedent, namely *the bird*: *the bird [I found the bird's nest]*, preserving the possessive marker from *whose*. Similarly in a question, we get: *which book did you read ( )* regularized as *did you read which book?*. To summarize, wh-expressions are introduced by a phrase containing a wh-word; following a wh-expression, there must be a gap, and this gap is understood as the wh-expression, after it has had the antecedent of the wh-word word filled in (if in a relative clause).

[1]The expression *replace by its antecedent* is used loosely here. What is really meant is replacing the relative pronoun by a pointer to the antecedent. This preserves co-referentiality of the relative pronoun and its antecedent, and avoids the dangers of copying quantifier and other modifier information.

The need for a gap can be captured very simply by associating with each grammar definition a set of paired input/output parameters. The input parameter signals whether or not a gap is need when the node is about to be constructed, at rule invocation time. The output parameter signals whether that need has been satisfied once the node is completed, at rule exit. Thus an assertion in a relative clause has as its input parameter the need for a gap (**need_gap**) and on exit, that need must have been satisfied (indicated by a **no_gap** output parameter). These parameters, once set, are simply passed along from parent to child, and sibling to sibling, via unification through linked input/output parameters.

However, an assertion may also occur as the main clause, where it is not licensed for a gap. This is illustrated in Figure 2 by the (simplified) definition for a sentence, as having two alternatives: an assertion or a question. The definition for *assertion* itself therefore must be neutral with respect to gaps, since that depends on where it is called from (relative clause or sentence). The parameters in the assertion definition simply pass along the information from parent to child and sibling to sibling. If the assertion is in a relative clause, then the need for a gap is passed along until some node (*nullwh* in Figure 2) realizes the gap (that is, accepts the empty string), at which point its output parameter is set to **no_gap**; this is passed along and finally, back up to assertion. If the assertion occurs as the main clause of a sentence, it has no need for a gap and in fact, cannot unify with the gap realization rule, which requires an input parameter of **need_gap**.

This mechanism enforces the constraint that only a node with the parameter pair (**need_gap/no_gap**) can dominate a gap. Any node whose input and output parameters are equal has not 'changed state' – that is, whatever it needed (or didn't need) on rule entry, it will still need at rule exit. Procedurally, any rule whose input and output parameters are equal cannot unify with the gap realization rule. The flow of information through the tree is illustrated in Figure 3.

```
% Simplified BNF definitions before parameterization for wh-constructions:

sentence     ::= assertion; question.
rel_clause   ::= wh, assertion.
assertion    ::= subject, verb, object.
subject      ::= noun_phrase.
verb         ::= *v.                  % * indicates terminal lexical category
object       ::= noun_phrase; assertion;....

noun_phrase ::= lnr; *pro.
noun_phrase ::= nullwh.
lnr          ::= ln, *n, rn.          % noun with left, right adjuncts.
rn           ::= null; pp; rel_clause.

null         ::= ' .                  % empty string (for empty adjunct slots)
nullwh       ::= ' .                  % empty string for gap realization.

wh           ::= [who]; [which]; ....

% Parameterized BNF definitions for handling relative clause:
% Where parameters pass no information, input = output parameter.

sentence(X/X)       ::= assertion(no_gap/no_gap); question(need_gap/no_gap).
rel_clause(X/X)     ::= wh(Y/Y), assertion(need_gap/no_gap).
assertion(In/Out)   ::= subject(In/Subj), verb(Subj/Verb), object(Verb/Out).
subject(In/Out)     ::= noun_phrase(In/Out).
verb(In/In)         ::= *v.
object(In/Out)      ::= noun_phrase(In/Out); assertion(In/Out); ...

noun_phrase(In/In) ::= lnr(In/In); *pro.
noun_phrase(need_gap/no_gap)
                    ::= *nullwh.                    % empty string for gap
lnr(In/In)          ::= ln(In/In), *n, rn(In/In).  % noun + left, right adjuncts
rn(In/In)           ::= null(In/In); pp(In/In); relative_clause(In/In).

null(In/In)         ::= ' . % empty string (for empty adjunct slots)
nullwh(need_gap/no_gap)
                    ::= ' . % empty string for gap realization.
wh(In/In) ::= [who]; [which]; ....
```

Figure 2: Simplified Rules with Parameters for Wh.

```
np(X/X)

n     rn(X/X)

      rel_clause(X/X)

      wh(Y/Y) assertion(need_gap/no_gap)

mice  which   subj(need_gap/need_gap) verb(need_gap/need_gap) obj(need_gap/no_gap)

              np(need_gap/need_gap)   v                       np(need_gap/no_gap)

              pro(need_gap/need_gap)  eat                     nullwh(need_gap/no_gap)

              they
```

Figure 3: Flow of Information in *...mice which they eat.*

1.3 The Framework: Restriction Grammar

The proposed solution is presented in the context of Restriction Grammar [8], which is the syntactic portion of the PUNDIT text processing system [9]. However, this solution is only dependent on a few general properties of Restriction Grammar, which it shares with other formalisms (e.g., Definite Clause Translation Grammars [1]). A Restriction Grammar is written in terms of context-free BNF definitions, augmented with constraints (*restrictions*) on the well-formedness of the resulting derivation tree. Constraints operate on the derivation tree, which is constructed automatically during parsing; restrictions traverse and examine this tree, to determine well-formedness.

One of the significant characteristics of Restriction Grammar is the *absence* of parameters. Context sensitivity is enforced by the restrictions, which obtain information from the derivation (parse) tree, rather than via parameter passing. Restriction Grammar is implemented as a form of logic grammar which includes parameters not just for the word stream, as in DCG's, but also for the automatic construction of the derivation tree as well. In addition, each grammar rule is augmented with an associated regularization rule (indicated by a right hand arrow $\rightarrow$), which incrementally constructs an *Intermediate Syntactic Representation (ISR)*. The ISR is an operator/operand notation that represents a canonical, regularized form of the parse tree. The regularization rule composes the ISRs of the daughter nodes in the derivation tree into the ISR of the current node, using lambda reduction. Computation of the ISR for wh-constructions is discussed in greater detail in section 6.

1.4 The Solution: Meta-Rules

Although parameterization is an elegant and efficient solution, it presents a major problem – it obscures the declarative aspect of the BNF rules, and correct parameterization of rules can be tedious and error prone, especially since there are some 40 object types in our current broad-coverage grammar of English.

The solution is to define a set of annotations to express the required linguistic constraints: gap introduction via wh-word, gap realization, and gap prohibition. Figure 4 shows a grammar using annotations defined as prefix operators applied to the node names in BNF definitions. Gap introduction is written as <<, gap realization as >>, and gap prohibition as <>. These are used, for example, to flag the need for a wh-word in a wh-expression, followed by the need to realize a gap:

```
rel_clause ::= <<wh, >>assertion.
```

Annotations can appear on either the left-hand side or the right-hand side of BNF definitions. By introducing the gap-requirement on the right-hand side of a BNF definition, we create a conditional gap requirement. For example, *assertion* requires a gap in the context of a *relative clause*, but not as the normal realization of a sentence (main clause) option. Thus we do not want to annotate the definition for *assertion*, but the call to *assertion* in *rel_clause*. However, the definition for *nullwh* is always a gap realization rule, hence it is annotated on the left-hand side (see Figure 4).

In certain cases, we need to define a special gap-requirement rule. For example, we define a special case for noun-phrase gap realization. This enables us to *block* transmission of gap parameters in all other options of noun phrase. To do this, we use the third annotation <> to set input parameter equal to output parameters. This annotation is also used to show that the verb can never license a gap. Similarly, the determiner (*det*) and pre-nominal adjective (*adjs*) rules cannot license a gap.

The remainder of the rules require no annotation; their parameters simply transmit whatever gap information is passed in. Figure 5 shows the parameterized definitions corresponding to the annotated definitions used in Figure 4.

```
% ANNOTATED SOURCE RULES

% Operator definitions
:- op(500,fx,[<<,>>,<>]).

% Relative clause requires  gap in assertion.
rel_clause  ::= <<wh, >>assertion.

assertion   ::= subject, verb, object.
subject     ::= noun_phrase.
<>verb      ::= *v.          % verb can't have gap.
object      ::= noun_phrase; assertion..

% Regular noun phrase rule.
% Parameters block gap
<>noun_phrase ::= lnr.
% Gap realization rule
>>noun_phrase ::= nullwh.

lnr         ::= ln, *n, rn.  % lnr = left-adjunct + noun + right-adjunct

% Left noun adjunct rules
ln          ::=  det, adjs.
<>det       ::= *t; null.    % t = determiner
<>adjs      ::= null; *adj, adjs.

% Right noun adjunct rules
rn          ::= null; pp; rel_clause.
pp          ::= *prep, noun_phrase.

% Normal empty string rule
null        ::= ' .
% Gap realization rule
>>nullwh    ::= '.

% Wh-word Rules
<<wh        ::= [which]; [who].
```

Figure 4: Illustration of Annotated Rules for Wh

```
% PARAMETERIZED VERSION OF ANNOTATED RULES

% Operator definitions
:- op(500,fx,[<<,>>,<>]).

% rel_clause       ::= <<wh, >>assertion.
rel_clause(In/In)  ::= wh(need_wh/no_wh), assertion(need_gap/no_gap).

% Gap propagation rules, generated via Meta-Rule
assertion(In/Out)  ::=
        subject(In/Subj), verb(Subj/Verb), object(Verb/Out).
subject(In/Out)    ::= noun_phrase(In/Out).
verb(In/In)        ::= *v.                          % <>verb ::= *v.
object(In/Out)     ::= noun_phrase(In/Out);
                       assertion(In/Out).

%  Regular noun phrase rule.
%  Annotation blocks gap: <>noun_phrase ::= lnr.
noun_phrase(In/In) ::=         lnr(In/In).
% Gap realization rule: >>noun_phrase ::= nullwh.
noun_phrase(need_gap/no_gap)
                   ::= nullwh(need_gap/no_gap). % the empty string
lnr(In/Out)        ::= ln(In,Out1), *n, rn(Out1/Out).
% Left noun adjunct rules
ln(In/Out)         ::=  det(In/Out1), adjs(Out1/Out).
det(In/In)         ::= *t; null(In/In).          % <> det ::= *t; null.
adjs(In/In)        ::= null(In/In); *adj, adjs(In/In).
                                                 %<>adjs ::= null; *adj, adjs.
% Right noun adjunct rules
rn(In/Out)         ::= null(In/Out); pp(In/Out); rel_clause(In/Out).
pp(In/Out)         ::= *prep, noun_phrase(In/Out).
% Normal empty string rule
null(In/In)        ::= '' .
% Gap realization rule
nullwh(need_gap/no_gap) ::= ''.
% Wh-word Rules
wh(need_wh/no_wh)  ::= [which];[who].            % <<wh ::= [which];[who].
```

Figure 5: Illustration of Parameterized Rules for Wh

1.5 The Meta-Rule Component

The meta-rule component for parameterization is implemented as a general procedure which adds parameters to each production in the grammar. At grammar read-in time, each rule is parsed and parameterized appropriately, depending on its annotation. The basic case is no annotation, in which case the following rules apply (*Label* is the left-hand side of the BNF definition; *Rule* is the right-hand side):

```
% Basic case:
wh_params(Label,Rule,NewLabel,NewRule) :-
    check_head_params(Label,InParam/OutParam,NewLabel),
    take_apart(Rule,NewRule,InParam,OutParam),!.

check_head_params(Label,Params,NewHead) :-
        insert_param(Head,Params,NewHead).
insert_param(Head,Params,NewHead) :-
        NewHead =..[Head,Params].

% Conjunction
take_apart((A,B),(NewA,NewB),InParam,OutParam) :- !,
    take_apart(A,NewA,InParam,OutParamA),
    take_apart(B,NewB,OutParamA,OutParam).

% Non-Terminal
take_apart(Def,NewDef,InParam,OutParam) :-
    insert_param(Def,InParam/OutParam,NewDef),!.
```

If there is a terminal symbol, indicated by *, there is clearly no gap, and input and output parameters are equal; terminal symbols do not get parameterized:

```
% Terminal
take_apart(*Atom,*Atom,InParam,InParam) :- !.
```

Parameterization of embedded disjunction poses a problem, because there is a possibility that different disjuncts could instantiate the *In/Out* parameters differently. To catch this problem, each disjunct is computed separately, and a routine *same_params* checks the results, to make sure that they are consistent, before instantiating the parameters. If they are inconsistent, it issues a warning message. Otherwise, it unifies the inputs of the disjunctions; likewise, it unifies the outputs. It is always possible to avoid the warning message by splitting embedded disjunctions into separate rules, as was done for the noun_phrase definition (see Figure 4).

```
% Disjunction
take_apart((A;B),(NewA;NewB),InParam,OutParam) :-
    take_apart(A,NewA,InParamA,OutParamA),
    take_apart(B,NewB,InParamB,OutParamB),
    same_params((A;B),
        InParamA,InParamB,InParam,OutParamA,OutParamB,OutParam),!.
take_apart((A;B),_,_,_) :- !,
    print('$$$ Warning:  disjunct '), print(A), print(';'), print(B),
    print(' may not be parameterized correctly!').
```

There is also a special case for each annotation, for both the left-hand side of the rule (*check_head_params/3*) and the right-hand side (*take_apart/4*). For the case of the wh-word <<), the rule is shown below. In this case, the wh-word functions independently of other nodes in the definition and so it does not hook up to the input/output parameters.[2]

```
% << Needs wh-word
check_head_params(<<Head,(need_wh/no_wh),NewHead) :- !,
    insert_param(Head,(need_wh/no_wh),NewHead).

take_apart(<<Def,NewDef,InParam,InParam) :- !,
    insert_param(Def,(need_wh/no_wh),NewDef).
```

The gap-requirement annotation >> also disconnects the phrase containing the gap from its parent and siblings.

```
% >> Requires gap
check_head_params(>>Head,(need_gap/no_gap),NewHead) :- !,
    insert_param(Head,(need_gap/no_gap),NewHead).

take_apart(>>Def,NewDef,InParam,InParam) :-!,
    insert_param(Def,(need_gap/no_gap),NewDef).
```

Finally, the annotation <> enforces sameness of input and output, precluding realization of a gap:

```
% <> Rules out gap
check_head_params(<>Head,In/In,NewHead) :- !,
    insert_param(Head,In/In,NewHead).

take_apart(<>Def,NewDef,InParam,InParam) :- !,
    insert_param(Def,InParam/InParam,NewDef).
```

[2]In actuality, the wh-expression needs to pass some information about the wh-word to the expression containing the gap, and this rule will be revised in section 6.

This small set of annotations is sufficient to describe the wh-constructions in English with one minor addition, needed to handle conjunction correctly (see section 7). Using this small set of annotations, the grammar writer can control the flow of gap information, without having a grammar cluttered with parameters. Unification is used to control generation of gaps only where required, so the technique is also efficient, avoiding extensive search to determine presence/absence of a gap.

1.6 Refinements

The treatment described above leaves open several issues. The first of these is the proper generation of a regularized syntax (*Intermediate Syntactic Representation*) for these constructions. It is clear that the compositional representation of a gap-containing expression can readily be described as a lambda expression. First, the wh-expression itself can be viewed as a lambda expression, which produces a *filler* when applied to the referent of the relative pronoun. Then, the gap-containing expression is a lambda expression, which when applied to the filler, produces a completed clause:

the book which I bought ( ):

```
=>     book1, lambda(Filler, [I bought Filler]), [lambda(Wh,[Wh]),book1]
=>     book1, lambda(Filler, [I bought Filler]), [book1]
=>     book1, [I bought book1].
```

Only wh-words in relative clauses have antecedents, namely the head noun to which the relative clause is attached. Wh-words in questions stand for the questioned element in the clause; in this case, we insert a dummy element **wh_gap** to mark the questioned element.

what did you buy ( )?

```
=>    lambda(Filler,[you buy Filler]), [lambda(Wh, [Wh]), wh\_gap]
=>    lambda(Filler,[you buy Filler]), [wh_gap]
=>    [you buy wh_gap]
```

This treatment extends very nicely to complex wh-expressions, such as *the person whose book I borrowed*, which can be represented as follows:

the person whose book I borrowed ( )

```
=>    person1,
        lambda(Filler,[I borrow Filler]), [lambda(Wh, [Wh 's book]), person1]
```

```
=>    person1, lambda(Filler, [I borrow Filler]), [person1 's book]
=>    person1, [I borrow person1's book]
```

This treatment has one unfortunate property: at the time that the lambda variable is generated for the relative clause, there is no way of knowing where it should be placed in the lambda expression, that is, where the associated gap in the assertion will be. For example, the lambda variable associated with a noun phrase gap could be realized as the subject or the object of the clause. Our solution is to pass the lambda variable along, embedded in the gap parameters, until the noun-phrase gap-realization rule rule is reached, at which point the lambda variable becomes the representation of the gap, shown in Figure 6.

We implement this by treating **need_gap** as a functor with an argument for the lambda variable. To avoid explicit mention of parameters in the source rules, we again use an annotation. Access to the lambda variable embedded in the **need_gap** parameter is given by the annotation **//lambdaVar(Var)**, where **//** is a binary operator.

Figure 6 shows several wh-constructions, with their associated ISR rules. The ISR rule appears on the right hand side of the arrow '->'. The node names within the ISR rule access the ISRs associated with the named node.

There are other complications in covering wh-constructions. One issue is that several different types of gap can occur, specifically noun phrase gaps, adverbial phrase gaps, and even adjective gaps:

The book that I bought (np gap)
What did you buy? (np gap)
The place I put it (adverb gap)
Where did you get it? (adverb gap)
The way I did it (adverb gap)
How big is it? (adjective gap)

In many cases, the wh-word signals the type of gap to be expected. This information is critical to determining what element should be realized as the gap. A simple way of handling this is to add a 'gap-type' argument to the information being passed in the parameters. Thus for *noun_phrase* to realize a gap, the gap type must be *np*; for an *adverb* to realize the gap, we must have gap type *adv*. This information is computed in the handling of the wh-expression and passed along to the gap-licensing construction. Thus the wh-expression does in fact have to communicate information to the construction dominating the gap. This is implemented by a slight complication to the meta-rule. The gap type information is accessed by the annotation **//type(GapType)** on the appropriate rule.

Finally, the semantics needs to know what the wh-word word was, so that it can distinguish between location expressions (*the place where I left it*) from temporal expressions (*the month . left*). Of course, there is not always an overt

```
% ANNOTATED RULES with ISR

:- op(500,fx,[<<,>>,<>,><]).
:- op(400,xfy,//).

<>rn      ::=  null -> lambda(N, [N]);
               pn -> lambda(N, [N, !pn]);
               rel_clause
                    -> lambda(N, [N, [-rel_clause, [+rel_clause, copy(N)]]]).
% +Def extracts the head of a list; -Def get the tail of the list.
rel_clause ::= <<wh, >>assertion//lambdaVar(Gap)
                        -> [wh, lambda(Gap, [assertion])].

assertion ::=  subject, verb, object -> [verb, object, subject].
subject   ::=  noun_phrase -> noun_phrase.
<>verb    ::=  *v -> lambda(Obj, [lambda(Subj, [v, Subj, Obj])]).
object    ::=  noun_phrase -> noun_phrase; assertion -> assertion.
<>noun_phrase
          ::= lnr -> lnr.
% gapped noun_phrase -- lambda variable is ISR.
>>noun_phrase//lambdaVar(Gap)
          ::= nullwh -> Gap.
<<wh      ::= [which];[who] -> lambda(Filler, [Filler]).

% PARAMETERIZED VERSION
rn(In/In) ::= null(In/In) -> lambda(N, [N]).
              pn(In/In) -> lambda(N, [N,!pn]);
              rel_clause(In/In)
                   -> lambda(N, [N, [-rel_clause, [+rel_clause, copy(N)]]]).
% +Def extracts the head of a list; -Def gets the tail of the list.
rel_clause(In/In)
          ::= wh(need_wh/no_wh), assertion(need_gap(Gap)/no_gap)
                   -> [wh, lambda(Gap, [assertion])].
noun_phrase(In/In)
          ::= lnr(In/In) -> lnr.
noun_phrase(need_gap(Gap)/no_gap)
          ::= nullwh(need_gap(Gap)/no_gap) -> Gap.
% omitted defs for assertion, subject, verb, object.
wh(need_wh/no_wh)
          ::= [which];[who] -> lambda(Filler, [Filler]).
```

Figure 6: The ISR Rules for Wh-Constructions

wh-word to indicate gap type: *the time I spent* (np gap) vs. *the time I visited* (adverb gap). All of these complications can be handled by embedding an additional parameter to carry the actual wh-word (if present) into the basic *need_gap* expression. This information is accessed by the annotation **//wh_word(WH)** and is used in constructing the final ISR.

1.7 Wh and Conjunction

A major complication occurs in the interaction between wh-constructions and conjunction. For example, with conjunction, it is not longer true that there is exactly one gap per wh-expression. Within a conjoined construction, the conjuncts must be identical with respect to gaps – if one conjunct has a gap, the other must have one as well, as in:

the books which I bought ( ) and you read ( )

or

the letter that ( ) arrived yesterday and I sent ( ) on to you.

but not

**the books which I bought and you read them*

In an interpreter-based treatment of wh-constructions, the interpreter needs to make a special provision for 'resetting' its state to handle gaps over conjoined structure – that is, it must account for the fact that there may be two gaps if the wh-construction has scope over a conjoined structure. One advantage of a meta-rule treatment is that this interaction occurs in an extremely natural way: the meta-rules for wh-expressions are applied first; then the conjunction meta-rules simply copy the parameters of the first conjunct for the second conjunct. Since these are implemented as logical variables, the parameters of the two conjuncts are unified, and hence are guaranteed to have exactly the desired behavior.

The ability to factor syntax and semantics cleanly makes it possible to implement a simple meta-rule treatment of conjunction [7]. The basic idea is to use meta-rules to generate all possible conjoinings as explicit rules. Because there are no parameters in the rules, and because the ISR rules are compositional and cleanly factored from the BNF definitions, the meta-rules are very simple. Figure 7 shows the transformation of a (simplified) assertion definition into a conjoined assertion definition. The resultant rule is a disjunction; one branch allows a conjunction, followed by a recursive call to assertion; the other branch terminates after application of a restriction. Each branch has associated with it a separate regularization rule. The original meta-rules, though simple, are low-level operations, concerned with maintaining and updating a recorded database of rules. Recently, a general mechanism to support the statement of meta-rules has been proposed [2] this treatment would provide a more elegant statement of the meta-rules used to handle conjunction in Restriction Grammar.

```
% Conjunction meta-rule (using '=>' as infix operator):
    (LHS ::= Body -> ISR) =>
    (LHS ::= Body,
        (*conj_wd, LHS  -> [conj_wd, ISR, LHS]
        ; {wconj_restr} -> ISR)).

% Example of meta-rule applied to assertion definition:

assertion ::= subject, verb, object -> [verb, subject, object].
        =>
assertion ::= subject, verb, object,
                (*conj_wd, assertion
                    -> [conj_wd, [verb, subject, object], assertion]
                ; {w_conj_restr} -> [verb, subject, object]).
```

Figure 7: Generation of Conjunction via Meta-Rule

The current conjunction meta-rules will handle correctly the interaction of wh-expressions and conjunction. However, one minor problem is the preservation of a source form of the rules, for inspection and editing by the grammar-writer. This requires that conjunction operate on the source (unparameterized) form of the rules. By introduction of one addition annotation ><, it is possible to apply conjunction to the source rule, which can then be parameterized via the wh meta-rule component. The >< annotation simply ensures that its operand receives the parameters associated with the left-hand side of the definition. Thus the conjunction meta-rule generates:

```
assertion ::= subject, verb, object,
                ( *conj_wd, ><assertion
                    -> [conj_wd, [verb, subject, object], assertion]
                ; {w_conj_restr} -> [verb, subject, object]).
```

The introduction of the >< annotation requires a slight complication to the code for **take_apart**, but is very straight-forward. The result is the following parameterized definition for assertion:

```
assertion(In/Out) ::=
     subject(In/Subj), verb(Subj/Verb), object(Verb/Out),
        (*conj_wd, assertion(In/Out)
              -> [conj_wd, [verb, subject, object], assertion]
        ; {w_conj_restr} -> [verb, subject, object]).
```

Combining treatment of conjunction and wh-constructions, the system is able to parse sentences such as:

The disk which he repaired and she installed is working.
The disk which she repaired and installed has failed.
What disk did she repair and he install?
The disk which was installed but not repaired has been removed.
Which disks and drives are failing?
What does she believe they will repair and the engineer will maintain?

1.8 Conclusion

It is clear that meta-rules provide a powerful approach to a range of grammatical problems. In particular, meta-rules are very well-suited to the handling of phenomena that require a regular change to a large range of grammatical constructions. Although the parameterization approach outlined here for wh-constructions differs considerably from the meta-rule treatment of conjunction, they both share the property of enabling the grammar writer to capture a high-level generalization that applies to many rules in the grammar. Perhaps even more interesting is the fact that these meta-rule treatments appear to combine gracefully, in a way that does not appear to be readily available through the extended interpreter approach of Extraposition Grammar.

We plan to investigate other possible applications of parameterized grammar rules. These include the use of parameters to propagate feature information, and the use of parameters to 'compile' restrictions into unification of parameters, for greater efficiency and greater *locality* within subtrees. By using parameters instead of explicit constraints on tree-shape, it may be possible to save well-formed subtrees. The problem of putting together well-formed subtrees can then be captured via unification, rather than by constraints which must look outside the local subtree.

As we continue to develop meta-rule approaches to various grammatical phenomena, it will be important to abstract from the specifics outlined here and move towards a more general language for the statement of grammatical meta-rules. Just as grammar examples provide fertile ground for the application of

meta-programming techniques, we expect meta-programming techniques to provide more elegant and efficient ways of capturing the meta-rules.

1.9 Acknowledgements

A number of people have made important contributions to this work. The basic grammatical insights come from Marcia Linebarger, the principal developer of our broad-coverage English grammar. The implementation of the ISR has been done by John Dowding, who also suggested the appropriate treatment of gaps and of complex wh-expressions. Deborah Dahl, Rebecca Passonneau and François Lang provided a number of helpful suggestions on the organization of the paper. I am also indebted to Dale Miller for interesting insights about the meta-programming aspects of the wh-problem.

[1] Harvey Abramson. Definite clause translation grammar. In *Proc. 1984 International Symposium on Logic Programming*, pages 233–241, Atlantic City, NJ, 1984.

[2] Harvey Abramson. Metarules and an approach to conjunction in definite clause translation grammars: some aspects of grammatical metaprogramming. In *Logic Programming: Proc. of the 5th International Conf. and Symposium*, pages 233–248, MIT Press, Cambridge, MA, 1988.

[3] Amelie Banks and Manny Rayner. Comparatives in logic grammars – two viewpoints. In *Proc. of the 2nd International Workshop on Natural Language Understanding*, Simon Fraser University, Vancouver, B.C, August, 1987.

[4] Veronica Dahl and Michael McCord. Treating co-ordination in logic grammars. *American Journal of Computational Linguistics*, 9(2):69–91, 1983.

[5] John Dowding and Lynette Hirschman. Dynamic translation for rule pruning in restriction grammar. In *Proceedings of the 2nd International Workshop On Natural Language Understanding and Logic Programming*, Vancouver, B.C., Canada, 1987.

[6] G. Gazdar. Unbounded dependencies and co-ordinate structure. *Linguistic Inquiry*, 12:155–184, 1981.

[7] Lynette Hirschman. Conjunction in meta-restriction grammar. *Journal of Logic Programming*, 4:299–328, 1986.

[3] Lynette Hirschman and Karl Puder. Restriction grammar: a prolog implementation. In D.H.D. Warren and M. VanCaneghem, editors, *Logic Programming and its Applications*, pages 244–261, Ablex Publishing Corp., Norwood, N.J., 1986.

[9] Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Schiffman] Passonneau, Lynette Hirschman, Marcia Linebarger, and John Dowding. Recovering implicit information. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, August 1986.

[10] F.C.N. Pereira. Extraposition grammars. *American Journal of Computational Linguistics*, 7:243–256, 1981.

[11] Fernando Pereira and Stuart Shieber. *Prolog and Natural-Language Analysis*. University of Chicago Press, Chicago, Illinois, 1987.

[12] C. Raze. A computational treatment of coordinate conjunctions. *American Journal of Computational Linguistics*, 1976. Microfiche 52.

[13] Naomi Sager. Syntactic analysis of natural language. In *Advances in Computers*, pages 153–188, Academic Press, New York, NY, 1967.

[14] William A. Woods. Progress in natural language understanding: an application to lunar geology. In *AFIPS Conference Proceedings*, 1973.

1.10 Appendix

Meta-Rules for Parameterization of BNF Definitions

```
% Do parameter generation at rule read-in time, to capture meta-rules.
% Pick up rule, add parameters, and record new rule.
Label ::= Rule :-
  wh_params(Label,Rule,ParamLabel,ParamRule),
  record(ParamLabel,(ParamLabel ::= ParamRule),_).

% Generate wh-parameters
wh_params(Label,Rule,NewLabel,NewRule) :-
make_head_params(Label,InParam/OutParam,NewLabel),
take_apart(Rule,NewRule,InParam,OutParam).

% Parameterize head of rule
% Case 1 -- head cannot dominate gap; close off parameters
make_head_params(Head,InParam/InParam, NewHead) :-
cant_pass_on(Head),!,
insert_param(Head,InParam/InParam,NewHead).
% Case 2 -- parameterize head.
make_head_params(Head,InParam/OutParam, NewHead) :-
insert_param(Head,InParam/OutParam,NewHead).

% TAKE APART RULE BODY to insert params
% Semantics Rule
take_apart((A -> B), (NewA -> B), InParam,OutParam) :- !,
take_apart(A,NewA,InParam,OutParam).
% Conjunction
take_apart((A,B),(NewA,NewB),InParam,OutParam) :- !,
take_apart(A,NewA,InParam,OutParamA),
take_apart(B,NewB,OutParamA,OutParam).
% Disjunction
take_apart((A;B),(NewA;NewB),InParam,OutParam) :-
take_apart(A,NewA,InParamA,OutParamA),
take_apart(B,NewB,InParamB,OutParamB),
```

```
same_params((A;B),
InParamA,InParamB,InParam,OutParamA,OutParamB,OutParam),!.
take_apart((A;B),_,_,_) :- !,
print('$$$ Error:  disjunct '), print(A), print(';'), print(B),
print(' cannot be parameterized correctly!').
% Literal
take_apart([Word|MoreWords],[Word|MoreWords],InParam,InParam) :- !.
% Restriction
take_apart({Restr},{Restr},InParam,InParam) :- !.
% Prune
take_apart(prune(Name,Defs),prune(Name,NewDefs),InParam,OutParam) :- !,
take_apart(Defs,NewDefs,InParam,OutParam).
% Literal
take_apart(*Atom,*Atom,InParam,InParam) :- !.
% Empty node
take_apart(" , " , InParam,InParam) :- !.
% Non-Terminal
take_apart(Def,NewDef,InParam,OutParam) :-
check_params(Def,InParam,OutParam),
insert_param(Def,InParam/OutParam,NewDef),!.
take_apart(Def,Def,InParam,InParam).

% cant_pass_on sets two parameters equal in the head of the name rule;
% this means that the automatically generated nodes of this type
% CANNOT contain a gap or a wh word.  These nodes have hand-generated
% parameterized rules that permit gap or wh-words.
cant_pass_on(sa).
cant_pass_on(rn).
cant_pass_on(nstg).
cant_pass_on(thats).

insert_param(Head,X/Y,NewHead) :-
Head =.. [Head|Args],
NewHead =..[Head,X/Y|Args].
```

Hirschman, Lynette. "Natural Language Interfaces for Large Scale Information Processing". In *Integration of Information Systems: Bridging Heterogeneous Databases*, ed. A. Gupta, IEEE Press, New York, 1989, pp. 308-314.

Natural Language Interfaces for Large-Scale Information Processing

LYNETTE HIRSCHMAN

1. Introduction

THIS paper outlines the role and availability of natural language processing in managing large scale information processing applications. The use of natural, as opposed to formal, languages is a key ingredient in facilitating human-machine communication, where "natural language" is taken to mean use of either spoken or written language, by the human and by the machine. Use of natural language raises the machine nearer the human level, rather than expecting the human to accommodate to the level of the machine. The vision in the discussion that follows is of *intelligent machine assistants* aiding the human decision maker in assimilating, organizing, storing, accessing, and integrating vast amounts of data. Without this assistance, it is clear that decision making will be labor-intensive, uneven in quality, slow, expensive, and probably insufficiently informed for optimal decision-making.

To date, natural language processing (NLP) has had very limited impact on interfaces to large-scale applications. This technology is only now maturing to the point where commercial applications are feasible; we expect major breakthroughs in the next five to ten years in: 1) providing custom systems for natural language access to distributed information systems; 2) capture of information in natural language (NL) form for building information systems; 3) providing limited "toolkits" which can be customized by end users for limited NL interface applications; 4) combining NL techniques with other media: icons, menus, pointing devices, to provide succinct high-level interfaces; and 5) supporting communication via *spoken* language, especially speech recognition, but also speech generation, in addition to written language. Realistically, a robust broad-coverage speech system is not likely to be available for at least six to eight years. However, there are a number of important applications of NL that are likely to be available within a five-year horizon. This paper will describe the state of the art, and identify some key applications with near-term payoff.

Although the obvious applications of NLP are to user interfaces, there are some other important contributions that this area will provide. For example, one of the major issues emphasized in the Highlights of the Knowledge-Based Integrated Information Systems Engineering Project [11] is the ability to define a "global semantics for distributed databases. A study of how people talk about the data in such databases can provide the underpinnings to such a mapping. The semantics required to formulate an NL front-end may capture much of the global semantics required by such a system.

The specific issue of temporal processing is significant enough to merit special mention. In a dynamic situation, especially a decision support application, it is critical to be able to capture temporal information, retrieve it, reason about it, and talk about it. NL researchers have been among the chief groups addressing this issue, and their work can make major contributions to the processing of temporal relations. This paper will discuss some of the ongoing work in this area in Section 6.1.

2. Dimensions of Natural Language Processing

This section describes various dimensions of natural language processing applications. These dimensions can be summarized as follows:

1) *Directionality of communication,* human to computer (understanding) and computer to human (generation). In addition, although the focus is on natural language, we will discuss several "mixed" approaches, which rely on techniques other than NL techniques. In particular, Section 3.2 discusses combining information retrieval techniques with natural language techniques; and Section 5 discusses combining graphics, menu, and mouse interfaces with natural language.
2) *Written versus spoken language,* for both recognition and generation. Although spoken language generation has reached a reasonable level of sophistication, spoken language understanding remains a very difficult issue: speaker-independent continuous speech understanding with a nontrivial vocabulary size (2000+ words) is still several years away. One of the critical issues here will be the ability of a speech understanding system to adapt to the speech of different users. Many systems require extensive training for each new user; the training requirement for future systems will decrease, as research makes progress in ability to model a range of users and to provide rapid speaker adaptation.
3) *Size of domain,* ranging from a very narrow domain, covering very limited topics, to completely unconstrained discourse. To date, the major successes have been achieved by constraining the domain of discourse to a narrow, well-defined topic (e.g., newspaper stories about Cyrus Vance, database queries to a database with information on Fortune 500 companies, spoken text

Reprinted with permission from *First Symp. Knowledge-Based Integrated Info. Sys. Eng.*, Feb 1987

editing commands, etc.). A constrained domain limits vocabulary, limits lexical ambiguity, and limits the kind of things that can be talked about, making it possible to build a reasonably detailed domain model. As the domain expands or as multiple domains are covered, problems of domain modeling and lexical ambiguity can become severe. Although some systems attempt to cover an application area rather than a domain (e.g., checking business correspondence for grammatical correctness in EPISTLE [22], [14]), coverage of broad domains significantly increases the complexity of the problem.

4) *Complexity of utterance,* ranging from single sentence to complex multiparagraph discourse. The difficulty grows in complexity as the task moves from single sentence utterances requiring relatively little in the way of discourse processing machinery, to complex discourse, which requires an elaborate set of modules for processing discourse relations, including machinery to carry along discourse context, as well as modules for reference resolution, temporal reasoning, and reconstruction of contextually implicit information. This applies to both recognition and generation.

Application types fall into three basic classes, which cut across the dimensions enumerated above. The three classes are:

a) information access
b) information capture
c) mixed initiative interactions (human-machine dialogue).

The first class of application, *information access*, is what usually comes to mind when discussing natural language interfaces to data/knowledge bases. This is the application area where commercial products are becoming available (e.g., Intellect, Q&A, Themis). The key requirement for information access is the use of natural language understanding to translate a natural language request into a DB/knowledge base query. The minimum requirement is ability to understand single sentence utterances. A more sophisticated interface would add the capability to understand a series of queries, including the ability to handle pronouns and ellipsis, but such a system requires substantially less sophistication than a system for information capture (see discussion below). Natural language generation can be used to enhance intelligibility of the answers in certain situations [15], as well as to offer feedback and/or explanations.

Information capture emphasizes the ability to understand paragraph and multiparagraph input. This requires a substantially more sophisticated system than simple query processing. Information capture via natural language has been an active area of research for over fifteen years, and is rapidly maturing. One indication of this is the *Message Understanding Conference*, which was held in San Diego in 1987. The goal of that conference was to demonstrate and compare capabilities of various message processing systems, given a Navy domain of intelligence messages (RAINFORMS). Given the state of the technology, it seems likely that there will be systems routinely capturing natural language text input in limited domains within the next five to seven years (e.g., medical reports [12], equipment failure reports [19], [7], intelligence messages [1], and bank telex messages [18]).

The third area, *mixed initiative* or *dialogue interaction,* requires the most sophistication, since it is a more open-ended situation which requires that the machine adapt to a variety of user responses for help or information. It requires an ability to handle connected discourse (as does the information capture application); it requires an ability to generate coherent explanations for its answers; and, finally, it requires an ability to provide "cooperative responses" to a range of users, including users who become increasingly sophisticated as they gain familiarity with the system. This in turn requires the ability to model the users of the system, and to update these models over time.

3. Information Access via Natural Language

Accessing information (e.g., databases or knowledge bases) via natural language is the most widely known application of natural language, probably because it is the only one that has had some significant commercial distribution in the form of systems such as Intellect, Q&A, and Themis. In addition, there are several powerful research systems, including SRI's TEAM system [9], and BBN's IRUS system [34]. By allowing a user to write queries in English, rather than in a formal query language, the information in the DB becomes more easily accessible to a wider community of users.

Accessing information via natural language raises a host of issues and implementation questions:

Portability—Can the system be ported to a different DB? To a different DBMS? To a different hardware/software environment?

Coverage—What portion of the language does it cover? Is this portion enough for users to stay inside it? Do users have to be trained to stay inside the language subset?

Reliability—What mistakes does the system make? Can it reliably recognize its own failure to understand? How does it help the user to "fix" a request that it hasn't understood? How does the system handle something outside the DB domain?

Intelligibility—How does the system feed the information back to the user? Does it offer an explanation? Does it request additional information in cases of ambiguity? Does it catch false assumptions on the part of the user?

System Integration—How does the system fit into the overall data processing architecture? Does it require special hardware/software? Can it be integrated into other facilities (report generation tools, pointing devices, etc.)? Can the system be used to access distributed databases? Knowledge bases?

3.1. Requirement for Access to Distributed Database Systems

The current systems provide only a very small portion of the capabilities outlined above. In addition, for complex decision-making applications, it is not clear that access to individual databases in isolation, by whatever means, addresses the information processing problem. What is really required is intelligent access to heterogeneous distributed data management systems, since in many applications, data may be

distributed, for historical reasons, across a number of different systems; finding a solution will involve accessing different kinds of data from the different systems and aggregating the data (using some kind of inference or reasoning procedure), and finally displaying the answer(s) in an appropriate fashion. The ideal architecture would then be a powerful natural language capability interfacing to an expert distributed database system, which would be able to retrieve data from different DBs and reason about it.

Since a natural language DB interface must define a mapping between the database semantics and the semantics of a given language, it is also natural to look to natural language processing techniques to aid in defining a "global" semantic mapping. In particular, the kind of information that a natural language interface needs, namely the aggregation (part-whole) hierarchy and the specialization/generalization (is-a) hierarchy, are two important methods of organizing information in defining database semantics. Natural language and DB technology need to interact in this area, to improve tools for providing a generic semantic description of databases that can be used both for global schema definition and NL interfaces.

3.2. Browsing through Large Databases

For some applications, what is required is not retrieval of a specific set of facts, but the ability to "browse" through the database. This is particularly important when 1) the user does not know exactly what information is needed, and 2) the volume of data is very large, so that a query might retrieve unmanagable amounts of data. This would be typical, for example, of search using key words, where the user tends to formulate retrieval requests based on feedback from the system. For example, if a retrieval request to a database containing equipment failure reports for all reports about *tapes* retrieves several thousand entries, the user will want to narrow the search to *bad magnetic tapes*, to retrieve only those reports of direct interest. Alternatively, if an initial request for *mag tape* retrieves nothing, the user might want to generalize the query to a search for *bad tapes*, or possibly may rephrase the query entirely, to retrieve the desired data.

To date, there has been interest in natural language techniques as applied to information retrieval problems, but few successful systems (even research systems) based on any kind of linguistic processing have been realized. One successful natural language front-end to an information retrieval system is the NLM CITE system [5], which provides a natural language interface to MEDLINE, a major medical bibliographic information retrieval system.

Even more experimental are the systems used to encode information for use as index terms in retrieval. The RECONSIDER system [32] offers an interesting approach based on the *structured text* found in *Current Medical Information and Technology (CMIT)*. The techniques in RECONSIDER are not linguistic, but exploit the regularities in CMIT to create inverted files of co-occurring terms, used for retrieval of diseases associated with symptoms. Limited natural language techniques have also been applied to generation of subject indexes [33] and to chemical reaction databases [28].

Although augmenting information retrieval with natural language techniques has received some attention, it is still an underexplored area that could have significant near-term payoff. For example, one strategy might be to use key word search to retrieval sentence units containing a particular word, followed by natural language processing, to determine whether the key word is in the appropriate context. By getting feedback from the user about which occurrences were or were not of interest, it would be possible for the system to build a profile of "interesting" occurrences, such that it could screen progressively more occurrences of the key word on its own.

These questions provide an initial framework for the evaluation and comparison of the various natural language systems currently under development.

4. Information Capture via Natural Language Understanding

The second major area of application is information capture: the ability to use natural language processing techniques to convert free-form textual information into a database or knowledge base. Such an application is sketched in Section 4.3 of the Knowledge-Based Integrated Information Systems Development Methodologies Plan where NLP is used to process formal information model statements in the IDEF framework [3].

There is a long history of research in this area, although information capture is a substantially more difficult task than query processing, due to the greater syntactic variety, including (for messages) telegraphic style, the necessity of handling referring expressions, and the need to handle discourse phenomena, such as discourse coherence, time, reference resolution, and recovering contextually implicit information.

Some of the early efforts in the area of text processing were done by the Linguistic String Project at New York University, which investigated processing of various types of medical reports [29], [30], [8]. Successors to this work include a joint Unisys-NYU effort, focused on the processing of Navy maintenance reports (CASREPSs) [7], [25]. Other efforts in this area include Schank's research [31], the NOMAD system developed by Granger at UC Irvine [6], Lebowitz' *Researcher* system [17], and Logicon's MATRES system [23].

These systems all share some features, at least in terms of input/output specifications. Input is free text; output is a representation of the information in terms of (quantified) predicates with arguments, represented in terms of case-frames, e.g., a verb followed by its (thematic role) arguments, as in *repair(agent(engineer),patient(compressor))*. In systems that handle time, there is generally the concept of an event or state, with an associated predicate-argument structure and a time (span) [26].

The possible applications of such processed text are numerous. Just as natural language queries can be mapped into DB retrieval requests, predicate argument structures can be converted in DB update requests (although handling of quantification and temporal modifiers raises some issues here). Other approaches include summarization [19] and simulation, and generation of index terms for information retrieval.

5. Mixed Initiative Dialogue

The third (and most sophisticated) type of human-machine interaction is the "mixed initiative dialogue," where machine and human collaborate in a conversational mode to achieve a particular goal [10].

This kind of system can be seen as an outgrowth of other scenarios discussed earlier. For example, a truly cooperative response, even in something like a DB query setting, would consist of a user interacting with (a machine version of) an expert DB administrator, who would aid the user (and possibly instruct the user) in how to obtain the particular information desired.

To achieve this kind of interaction, a range of capabilities is needed. For example, text generation will clearly be involved as soon as there is any kind of "conversation." It will be particularly important in providing explanations or instructions to the user.

A key element of task-oriented dialogue is the notion of progression from some initial state to a final state where a given task has been completed. The system must maintain a model of the changes over time, as the task moves from initial state to completion. Somewhat independent of this, the user will be learning about the task, especially if the system provides explanation and instruction—thus the machine will need to maintain a model not only of the state of task completion, but of information already presented to the user—that is, a dynamic model of the state of the user.

The concept of user modeling is even more important if the primary purpose of the system is to aid or instruct a range of users. In this case, the system must infer from the user's questions (or perhaps from some preliminary information furnished explicitly by the user) what the user's state of expertise is. This will enable the system to furnish the most relevant amount and depth of information to the user.

Intermediate between a fully automated system that learns as it interacts with the user, and a "dumb" system that cannot adapt to the user, is the notion of *machine-aided* task execution. This has been the dominant paradigm, for example, in the machine-aided translation systems, but is also applicable to other areas, such as financial transactions, intelligence stations, automatic zip-code assignment, etc. In a machine-aided setting, the machine performs what tasks it can without human help, but recognizes a class of inputs as being outside the set of information supplied. In such a system, the NL system would process the inputs and notify the operator of those transactions that it cannot process. The system could supply the information required to identify the transaction, and characterize the parameters which it used to recognize this as information it could not process. The operator would then interact to complete the processing (or at least eliminate the problem), and the "new" information would be added to the knowledge base (if appropriate), further extending the operations performed by the system. The interface might use speech synthesis to alert the operator and explain the status, but could then resort to more classic interfaces to accept additional inputs.

In addition to the notion of "mixed initiative dialogue," it is important to mention mixed media interaction, where the system (and the user) select the medium most appropriate to the type of information. For example, the user may want to point to an area on a map (using the mouse), in order bring up a tabular display of data about the area on the map, and then ask a question, in natural language, about some of the data in the display. Similarly, the system might "answer" a question about the location of an object on a map by *highlighting* the location of the object on the map. This notion of "seamless multimedia" interfaces is an active area of research at several institutions (CMU, ISI, BBN).

6. Spin-Offs from Natural Language Research

Aside from the types of human-machine interaction explored in the earlier sections, there are a number of areas that have great relevance outside strict natural language applications. These areas deserve special mention, because the research here may have broad applicability even when no natural language interface is involved.

6.1. Temporal Processing

The first of these is the processing of temporal information. Time is obviously a highly visible and important part of communication through language. It is communicated through a variety of linguistic devices, including verb tense (*The bus left* vs. *The bus leaves*), aspect (*The bus has left* vs. *The bus left*), adverbial expressions (*The bus left on Sunday*), verb semantics (*The bus' departure preceded the loading of the cargo*), subordinate clauses (*The bus left after they arrived*), and other related devices, such as causal statements (*The bus stopped because it was out of gas*).

By exploring the ways in which time is expressed in natural language, linguists have uncovered a rich variety of temporal relations [2], [27]. The ability to represent these relations and to update a knowledge base as it changes over time are clearly issues with relevance not just to natural language processing, but to any system that tries to represent a dynamic situation. Change over time also has extremely important implications for the database world, although conventional DB systems do not allow any sophisticated reasoning about temporal relations between events. This will clearly be a major research area for knowledge representation and databases in the next few years.

6.2. User Modeling

The area of user modeling, although mentioned in connection with mixed initiative dialogue systems, clearly is not limited to natural language interaction. The notion of user modeling is essential to the successful design of human-machine interfaces.

There are two dimensions to user modeling: the modeling of a range of users of a system, in order to tailor responses to the appropriate type of user [16]; and the ability of a given user to learn over time, so that information once given need not be repeated over and over again: explanations may become more succinct, for example.

The first kind of user modeling, namely detecting the level of a user, is somewhat analogous to fault diagnosis—the system needs to "diagnose" the level of the user, based on limited (and sometimes inconsistent) information.

The second aspect of user modeling will need to model the change in the user's state over the course of the interaction. In particular, such a user model needs to maintain its context, where the context consists of previously exchanged information. This will be changing continuously over the course of the dialogue. Failure to incorporate at least limited user modeling will lead to systems that seem very "dumb" and repetitious, as well as inflexible or unfriendly.

6.3. Tools for Global Semantics

The ideal user interface to a distributed, heterogeneous database system would provide the user with a uniform view of the information contained in the system: a global DB semantics. The construction of such a global semantics is a complex task; it involves reconciling different views of the data, different implicit semantics, different units of measure, etc.

Since natural language interfaces must also model a kind of global semantics, which relates language constructs to general semantic classes and relationships in the domain, the tools developed for modeling semantics in natural language may well provide a useful "global semantics" for heterogeneous databases. One useful strategy may be to define two levels of semantics—a general semantics, used for supporting the natural language interface, and a second "global DB semantics," which is defined by a mapping from the general concepts to those concepts and relationships supported by specific distributed databases. The relationship between the semantics needed for natural language and the kind of semantics needed for a global DB semantics is an interesting open research issue.

6.4. Knowledge Acquisition

At the heart of both expert systems applications and natural language applications is the issue of knowledge acquisition: how to get the domain-specific knowledge into the system. It is clear that advances in this area will also affect the DB area, since the same techniques used in expert systems can be used to structure databases, and to input information into databases. Section 4 discussed information capture using natural language techniques, but based on the assumption that one already had a natural language system running in a given domain. Here the issue is: how to get that system running.

For natural language systems, it is necessary to acquire both linguistic and general domain knowledge. There is extensive research now on building a range of tools to aid in this process. The first level of tool is a knowledge representation framework that permits the user to model various "second order" relationships in a domain, e.g., part-whole, and type relationships. A second level tool would prompt the user to generalize this information, and would check for missing and inconsistent information. More advanced tools might perform generalizations and analogies on their own, given novel problems and sets of data, to provide limited "machine learning." The success of expert systems technology long-term (and also natural language interfaces) will depend heavily on the availability of such tools to ease the knowledge acquisition process.

7. Conclusion

The preceding sections have discussed a variety of uses for natural language interfaces, as well as some tools that continued research in this area will make available. This section will provide a brief discussion of the timetable for such results.

7.1. Information Access

There are now commercial systems providing limited natural language access to databases. The language coverage is not very broad, and the systems are limited in their ability to flag and explain their errors. However, this technology is currently available and will continue to improve. It is clear that natural language interfaces have not created a sudden demand for more such systems. This is because typing natural language into a system, though perhaps more friendly for the naive user than a DB retrieval request, is still cumbersome. For natural language interfaces to become truly successful, it will be necessary to provide speech recognition, so that a user can *speak*, rather than type, to get information out. Speech recognition is still a very fragile technology, though enormous gains have been made in signal processing hardware and software. It seems likely that as researchers link the signal processing back-end with the linguistic/semantic frameworks developed for text processing, we should begin to see reasonable continuous speech recognition (in limited domains) in the next 4–8 years.

7.2. Information Capture

There are only very limited "custom systems" available today for information capture from natural language (for example, the bank telex system ATRANS, developed by Cognitive Systems [18]). However, this technology is maturing and over the next few years, there should be a number of custom systems developed to capture information from various types of messages. We are also seeing the first natural language "toolkit," namely Carnegie Group's LanguageCraft. Although LanguageCraft leaves much to be desired, it is an interesting first step in providing users with a limited tool set from which to build natural language interfaces (this is mostly aimed at building query interfaces, but has been used to build limited text processing interfaces as well—as mentioned in the Knowledge-Based Integrated Information Systems Development Methodologies Plan, Section 4.3 [3]).

For this class of application, there are still two serious problems: coverage of the system, and the issue of acquiring sufficient domain-specific knowledge. In general, most current systems have a fairly limited coverage of the language; those systems which have a broader coverage have problems with robustness and maintainability, particularly if the system is to be maintained by nonlinguists. The portability issue is being addressed in a number of ongoing research efforts, including work by Hirschman [13], Ballard [4], Moser [24], among others. However, tools in this area are still quite primitive and require substantial expertise on the part of the user. It will still be three to five years before we see good sets of tools to support portability of large scale systems.

It seems likely that we will see a steady growth in custom applications and an increasing number of "toolkits" over the next three to five years. In this area, there is less of a need to couple the system to speech recognition, since there are huge volumes of written material routinely transmitted; however, the ability to accurately capture spoken language would revolutionize various applications. For example, the introduction of a "talking typewriter," which transcribes spoken input, would have an significant impact on the office automation area. The commercial market here is enormous, and we are already seeing the first attempts at introducing such products.

7.3. *Mixed Initiative Dialogue*

The mixed initiative dialogue places the greatest demand on the system, to not only understand, but to generate reasonable responses. In the past, generation has been handled largely by canned or template-driven responses. However, there is substantial research going on in text generation, and there are now research systems capable of doing generation for specific tasks [21], [20]. However, these systems are still research systems, and it would require substantial work to turn them into a commercially marketable "toolkit." The obvious next step is to couple generation systems with speech synthesis systems to provide spoken output; these advances will be taking place in the next year or two.

The area of mixed initiative dialogue will prove to be an extremely useful one as expert systems applications become more widespread. Such systems will also play a major role in computer-aided instruction. Again, the addition of speech recognition and generation will probably make an enormous difference in user acceptance and utility of such systems.

In general the next five years to seven years should prove to be a pivotal period in the deployment of natural language interfaces. As speech technology develops to the point where spoken language can become the medium of interaction, the demand for such interfaces will probably increase dramatically. It is clear that the technology is maturing to the point where it will be able to provide order of magnitude improvements in the human/machine interface, and any system of the future will almost certainly include the ability to interact with the system via natural language, both written and spoken.

References

[1] Meyers, A., "VOX—An extensible natural language processor," in *Proc. IJCAI-85*, Los Angeles, CA, 1985, pp. 821–825.

[2] "Maintaining knowledge about temporal intervals," *Comm. ACM*, vol. 26, no. 11, 1983, pp. 832–843.

[3] *Knowledge-Based Integrated Information Systems Development Methodologies Plan*, A. Gupta and S. Madnick, Eds., Sloan School of Management, MIT, Cambridge, 1987. (NTIS and DTIC Accession Number 195851).

[4] Ballard, B., "TELI," in *Proc. 24th Ann. Conf. Assoc. Computational Linguistics*, 1986.

[5] Doszkocs, T. E. and B. A. Rapp, "Searching MEDLINE in English: A prototype user interface with natural language query, ranked output, and relevance feedback," in *Proc. ASIS Ann. Meeting*, Knowledge Industry Publications, 1979, pp. 131–139.

[6] Granger, R. H., C. J. Staros, G. B. Taylor, and R. Yoshii, "Scruffy text understanding: design and implementation of the NOMAD system," in *Proc. Conf. Applied Natural Language Processing*, 1983, pp. 104–106.

[7] Grishman, R. and L. Hirschman, "PROTEUS and PUNDIT: research in text understanding," *Computational Linguistics*, vol. 12, no. 2, pp. 141–145.

[8] ——, "Question-answering from natural language medical data bases," *Artificial Intelligence*, vol. 11, 1978, pp. 25–43.

[9] Grosz, B., "TEAM: A transportable natural-language interface system," in *Proc. Conf. on Applied Natural Language Processing*, Santa Monica, CA, Feb., 1983, pp. 39–45.

[10] ——, "Focusing and description in natural language dialogues," in *Elements of Discourse Understanding*, A. Joshi, B. Webber, and I. Sag, Eds., Cambridge, MA: Cambridge University Press, 1981, pp. 84–105.

[11] Gupta, A. and S. Madnick, *Knowledge-Based Integrated Information Systems Engineering: Highlights and Bibliography*, Sloan School of Management, MIT, Cambridge, MA, 1987. (NTIS and DTIC Accession Number A195850).

[12] Hirschman, L., G. Story, E. Marsh, M. Lyman, and N. Sager, "An experiment in automated health care evaluation from narrative medical records," *Computers and Biomedical Res.*, vol. 14, 1981, pp. 447–463.

[13] Hirschman, L., "Discovering sublanguage structures," in *Sublanguage: Description and Processing*, R. Kittredge and R. Grishman Eds. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1986.

[14] Jensen, K., G. E. Heidorn, L. A. Miller, and Y. Ravin, "Parse fitting and prose fixing: getting a hold on ill-formedness," *Computational Linguistics*, vol. 9, nos. 3–4, 1983, pp. 147–160.

[15] Kalita, J. K., M. L. Jones, and G. I. McCalla, "Summarizing natural language database responses," *Computational Linguistics*, vol. 12, no. 2, 1986, pp. 107–124.

[16] Kass, R. and T. Finin, "Rules for the implicit acquisition of knowledge about the user," in *Proc. of AAAI-87*, Seattle, WA, 1987.

[17] Lebowitz, M., "Researcher: An experimental intelligent information system," in *Proc. Ninth Int. Joint Conf. Artificial Intelligence*, 1985, pp. 858–862.

[18] Lytinen, S. and A. Gershman, "ATRANS: Automatic processing of money transfer messages," in *Proc. of AAAI-86*, Philadelphia, PA, 1986, pp. 1083–1088.

[19] Marsh, E., H. Hamburger, and R. Grishman, "A production rule system for message summarization," in *Proc. 1984 National Conf. on Artificial Intelligence*, Oakland University, Rochester, MI, 1984.

[20] McDonald, D. D. and J. Pustejovsky, "Tags as a grammatical formalism for generation," in *Proc. 23rd Annual Meeting ACL*, Chicago, IL, 1985, pp. 94–103.

[21] McKeown, K. R., *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language*, Cambridge, MA: Cambridge University Press, 1985.

[22] Miller, L. A., G. E. Heidorn, and K. Jensen, "Text-critiquing with the EPISTLE system: An author's aid to better syntax," in *Proc. Nat. Comp. Conf.*, AFIPS Press, Arlington, VA, 1981, pp. 649–655.

[23] Montgomery, C. A., "Distinguishing fact from opinion and events from meta-events," in *Proc. Conf. Applied Natural Language Processing*, Santa Monica, CA, Feb. 1983, pp. 55–61.

[24] Moser, M. G., "Domain dependent semantic acquisition," in *Proc. First Conf. Artificial Intelligence Applications*, Denver, CO, 1984, pp. 13–18.

[25] Palmer, M. S., D. A. Dahl, R. J. [Passonneau] Schiffman, L. Hirschman, M. Linebarger, and J. Dowding, "Recovering implicit information," presented at Proc. of the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, NY, Aug. 1986.

[26] Passonneau, R. J., "Situations and intervals," in *Proc. 25th Annual Meeting of the Assoc. for Computational Linguistics*, Stanford, July, 1987, pp. 16–24.

[27] ——, "A computational model of the semantics of tense and aspect," *J. Computational Linguistics*, vol. 14, June 1988, pp. 44–60.

[28] Reeker, L. M., E. M. Zamora, and P. E. Blower, "Specialized information extraction: Automatic chemical reaction coding from English descriptions," in *Proc. Conf. on Applied Natural Language Processing*, Santa Monica, CA, Feb. 1983, pp. 109–116.

[29] Sager, N., "Natural language information formatting: The automatic conversion of texts to a structured data base," in *Advances in Computers*, M. C. Yovits and M. Rubinoff, Eds. New York, NY: Academic Press, 1978, pp. 89–162.

[30] ——, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*, Reading, MA: Addison-Wesley, 1981.

[31] Schank, R. C., J. L. Kolodner, and G. DeJong, "Conceptual

information retrieval," in *Information Retrieval Research*, R. N. Oddy *et al.* Ed. London: Butterworth and Co. Ltd., 1981, pp. 94–116.

[32] Tuttle, M. S., D. D. Sherertz, M. S. Blois, and S. Nelson, "Expertness" form structured text? RECONSIDER: A diagnostic prompting program," in *Proc. Conf. on Applied Natural Language Processing*, 1983, pp. 124–131.

[33] Vladutz, G., "Natural language text segmentation techniques applied to the automatic compilation of printed subject indexes and for online database access," in *Proc. Conf. on Applied Natural Language Processing*, 1983, pp. 136–142.

[34] Weischedel, R., *et al.*, "BBN laboratories: research and development in natural language processing in the strategic computing program," *Computational Linguistics*, vol. 12, no. 2, 1986, pp. 132–136.

Hirschman, L., and Dowding, J. "Restriction Grammar: A Logic Grammar", to appear in *Logic and Logic Grammars for Language Processing*,, edited by Stan Szpakowicz and Patrick St. Dizier, Ellis Horwood, LTD.

Restriction Grammar: A Logic Grammar

Lynette Hirschman and John Dowding

Paoli Research Center
UNISYS Defense Systems
P.O. Box 517, Paoli, PA 19301
Telephone: (215) 648-7554
E-Mail: hirsch@prc.unisys.com

ABSTRACT

This paper describes the Restriction Grammar formalism and its use in a broad-coverage English grammar for the PUNDIT natural language processing system. Restriction Grammar is a logic grammar and contains features common to the logic grammar paradigm, e.g., a context-free grammar "backbone" augmented with constraints (restrictions). However, RG has a number of distinctive features which make it especially well-suited to the development of large-scale, compact, efficient and maintainable grammars. A distinguishing feature is the notion of restrictions: restrictions are well-formedness constraints on the parse (proof) tree. This limits restrictions in power: restrictions can only accept or reject structures; they cannot build structure. Restrictions obtain contextual information by traversing the parse tree, which acts as a kind of globally accessible data structure during parsing. The tree is passed as a parameter inserted uniformly at translation (or interpretation) time in all rules. As a result, no explicit parameters are needed in rules, and this, in turn, simplifies maintenance of the grammar. The absence of parameters makes it easy to formulate meta-rules to handle two difficult constructions, conjunction and wh-expressions. It also facilitates the coupling of grammar rules with regularization rules, so that a regularized syntactic structure is built up incrementally during parsing. Finally, constraints on the power of restrictions and the well-defined way in which non-local information is accessed make it possible to explore alternative parsing and control strategies. We are now actively investigating the use of delayed restriction execution, applied both to bottom-up parsing and to the use of well-formed substring tables. The RG formalism has also shown promising results for parallel processing (simulated at a 20-30 fold speed-up), based on the or-parallelism inherent in parsing with a broad-coverage grammar.

---

[1]This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research, and in part by Independent R&D funding from Unisys Defense Systems.

1. Restriction Grammar as a Logic Grammar

Restriction Grammar is a logic grammar framework that provides the syntactic component for PUNDIT, a natural language processing system developed over the past eight years at Unisys. Like most logic grammars, Restriction Grammar supports a grammar consisting of context-free rules (BNF definitions) and constraints or *restrictions*, to capture context-dependent relations. However, Restriction Grammar also has a number of interesting features that distinguish it from many logic grammars.

Restrictions: Restriction Grammar, like other logic grammars, is based on the notion of parsing as a proof of sentencehood. However, restrictions in Restriction Grammar can be viewed as well-formedness constraints on the structure of the parse tree (or proof tree). Restrictions gather contextual information by examining the parse, and not by using parameter passing. Also, restrictions are constrained in their power. In particular, they can only reject or accept proposed analyses; they do not **add** information or instantiate variables (in contrast to, for example, unification grammars). These features make it easy to write meta-rules in RG and also to experiment with alternative control strategies (parsing strategies).

Meta-Rules: RG uses meta-grammatical notions to capture linguistic generalizations and to maintain compactness of the grammar. A set of meta-rules generates rules for conjoined structures from definitions of simple (unconjoined) constituents. Wh-constructions are also treated via a different type of meta-rule. For wh-constructions, grammar annotations in the BNF definitions indicate the need for a gap or the filling of a gap. The annotated grammar rules are processed into executable form, which inserts paired parameters for passing information on the state of the gap.

Regularization Rules: Coupled with each BNF definition is a regularization rule for computing the Intermediate Syntactic Representation (ISR) compositionally from the surface syntax. The compositional computation of the ISR permits interleaving of syntax and semantics, to provide for earlier application of semantic constraints during parsing. For both conjunction and wh-constructions, the rules to generate the ISR integrate easily into the meta-grammatical framework.

Parsing Control: Restriction Grammar is implemented in Prolog. A translator turns RG rules (context-free BNF definitions with restrictions) into executable Horn Clauses in a fashion similar to DCGs. The parser for RG can be readily implemented as a top-down parser, and this has been the parsing strategy in use for most of our applications. One refinement has proved very useful, namely the *Dynamic Translator*, which mixes interpreted and translated code, to allow grammar rules to be modified during parsing. Because of the limited power of restrictions, we have also been able to use RG in a bottom-up, left-

corner parser. In this implementation, if a restriction requires more of the parse tree than has been built, it is delayed (by use of a *continuation*). We have also explored the notion of delayed restrictions and continuations for parsing with well-formed substrings.

Or-Parallelism: Parsing is a search problem. The form of RG has made it possible to exploit the or-parallelism inherent in the set of alternative grammar rules. Simulation experiments have indicated that speed-ups of 20-30 fold are possible, given a sufficient number of processors (40-60 processors). More recently, we have run a parsing application on a 12-processor system that showed as much as 11 fold speed-up.

In the following sections, we will discuss those features of Restriction Grammar that enable it to support a broad-coverage computational grammar of English, focusing on restrictions, the implementation of the ISR, and the implementation of the meta-treatments for conjunction and wh-constructions. We will then focus on the features of RG that support efficient parsing and that facilitate use of alternate search (parsing) strategies, including parallelism.

2. Background: Restriction Grammar as Used in PUNDIT

This section provides the context of our ongoing work on Restriction Grammar. We describe here the Restriction Grammar syntactic module in the context of the larger PUNDIT system. We also provide a quick overview of PUNDIT's grammar coverage, including a brief introduction to string grammar and a small sample Restriction Grammar.

2.1. The PUNDIT System

Restriction Grammar supports the syntactic component of the PUNDIT[2] natural language processing system that has been under development at Unisys over the past eight years [Hirschman1989]. PUNDIT has been demonstrated in a range of language understanding applications, e.g., for filling a database by processing text input; for querying a database; and for summarization of key information from text input. The system has been ported to a number of domains, including several types of Navy messages (maintenance reports, intelligence reports), medical abstracts, queries to a Navy resource management database, commercial computer failure reports, and transcriptions of air traffic control pilot/controller conversations. In addition to processing written text, we are now working on integrating PUNDIT with speech recognition technology, to support spoken language understanding.

PUNDIT is a modular system, whose components interact to process the syntactic, semantic and discourse information in a text, producing a meaning representation which forms the basis for the final, application-dependent output (database fill, database query, or summary). In addition to the syntactic

[2] Prolog UNDerstanding of Integrated Text.

component, PUNDIT has a semantic component [Palmer1986,Dahl1987] and pragmatic component[3] which contains modules for handling reference resolution [Dahl1986] and temporal information [Passonneau1988].

The syntactic module provides both a detailed analysis of the surface syntax of the sentence and a regularized or canonical representation, called the *Intermediate Syntactic Representation* or ISR. The ISR (described in detail in section 5) serves as the interface between syntax and semantics. Because the ISR is computed compositionally, it can support interleaving of syntactic and semantic constraints, which is critical to narrowing the search space associated with a large grammar.

To make use of selectional information during parsing[Lang1988], a restriction is called upon completion of each noun phrase node and each clause node. The restriction examines the ISR associated with the partial parse, extracting from the ISR the subject-verb-object relations (for clauses) or the host-modifier relations (for noun phrases). These relations are compared against lists of known semantically valid (or invalid) patterns that have been collected previously (by interaction with an expert in the subject area). On this basis, semantically valid phrases are allowed to continue; invalid ones are rejected (causing backtracking), and unknown phrases may (optionally) be sent to the user for a ruling on their validity.

The semantic component of PUNDIT generates a semantic representation of an input sentence by mapping the constituents of the ISR (operators and their arguments) into semantic predicates and their associated thematic roles. The semantics component is also interleaved with calls to pragmatics which determine noun phrase reference and assign temporal relations to the predicates.

2.2. String Grammar

A major motivation for the development of Restriction Grammar was a desire to use the broadest-coverage computational grammar available, namely Sager's String Grammar treatment of English [Sager1981]. Since certain string grammar notions play a role in the later discussion, this subsection provides a brief overview of key notions in string grammar, as formulated originally by Zellig Harris [Harris1962] and turned into a computational treatment by Sager and her co-workers [Grishman1973].

String grammar distinguishes two types of constructs: *head/adjunct* (endocentric) constructions and *strings* (exocentric) constructs. An endocentric construction has a *head* flanked by left modifiers and right modifiers; the behavior of an endocentric construction is governed by its head; that is, a noun phrase is noun-like in its behavior; an adjective phrase is "adjective-like", etc. The other important construction in string grammar is the *string*. A string is an

[3] See the chapter by Dahl and Ball in this volume

exocentric construction, that is, a construction whose behavior differs from that of its constituents. For example, an assertion cannot be considered "verb-like" or "subject-like" in its syntactic properties. A string is made up of two or more obligatory elements, plus optional adjuncts, e.g., in Figure 2 below, we see that the definition for `assertion` consists of the obligatory elements subject, tensed verb and object, interleaved with optional sentence adjunct slots.

We refer to endocentric constructions as `lxr` constructions, where `x` stands for the head, flanked by its left (`lx`) and right (`rx`) modifiers; the left and right adjunct slots may be empty. Figure 1 shows some of the important `lxr` constructions in string grammar; terminal nodes (lexical classes) are indicated by an asterisk. Note that the basic lexical classes (nouns, adjectives, tensed verb and verb participles) have associated `lxr` constructions. Left and right modifier definitions are shown for adjectives and pronouns.

```
% LXR Definitions
ltvr    ::= lv, *tv, rv.        % tv   = tensed verb
lvenr   ::= lv, *ven, rv.       % ven  = past participle of verb
lar     ::= la, *adj, ra.       % adj  = adjective
lpror   ::= lpro, *pro, rpro    % pro  = pronoun
lnr     ::= ln, *n, rn.         % n = noun

% Adjunct Definitions
la      ::= *adv; null.         % adv  = adverb
ra      ::= pn; null.           % pn   = preposition + noun
lpro    ::= *adv; null.         % e.g., "only I"
rpro    ::= *adv; null.         % e.g., "she alone"

sa      ::= pn; *adv; null.     % sa = sentence adjunct
```

Figure 1: Some `lxr` and Adjunct Rules in String Grammar

Strings include basic constructions such as assertion, question, and imperative. The prepositional phrase (pn) is also a string, since its behavior is neither that of a preposition nor that of a noun (in fact, it is often adverbial). Figure 2 is a (partial) list of strings.

```
assertion ::= sa, subject, sa, ltvr, sa, object, sa.
                                % sa = sentence adjunct
pn  ::= *p, nstg.               % prepositional phrase
veno ::= lvenr, sa, object.     % perfect, e.g, 'eaten a cookie"
tovo ::= to, lvr, sa, object.   % e.g, "to read a book"
```

Figure 2: Some String Rules in String Grammar

The philosophy of string grammar is to include a slot for each element; the realization of that element may be the empty string if the element is optional (e.g., adjuncts), or if it has been "zeroed" (reduced to zero or null) for some

other reason (e.g., gapping). The advantage to this approach is that the skeletal parse tree is very regular. For example, an assertion always contains nodes for subject, verb, and object, separated by sentence adjunct slots. However, many of those nodes may be empty (including the object, which can be realized as `nullobj` for an intransitive verb). Adherence to this philosophy reduces the number of grammar rules and makes for efficient top-down parsing, but also makes for bushy trees with many empty nodes.

2.3. Object Options in String Grammar

One large group of strings is the class of objects. String grammar handles auxiliaries as instances of verb + complex object. This gives a very regular, recursive structure to the object node in string grammar. At the top level, we have the tensed verb, followed by an object. If the tensed verb is a modal, its object will be `vo` – an infinitive followed by object, e.g., *I may read the book.* If the tensed verb is *have*, the object may be the past participle object `veno`, e.g., *They have been reliable*, etc. This means that objects carry a great deal of information, and may often contain a meaning-bearing verb where there are auxiliaries or a complex object (see Figure 4 below for an example).

2.4. Sample Restriction Grammar

A grammar in the RG framework is written as context-free rewrite rules, in the form of BNF definitions, augmented by *restrictions*, which are executed on the partial parse tree constructed up to that point. Restrictions are used to check for things such as agreement (subject-verb, determiner-adjective-noun), subcategorization (verbs can only take objects that they are subcategorized for), and positional constraints (certain adverbial expressions cannot occur between subject and verb). There are several types of restriction: (1) well-formedness restrictions, which check the well-formedness of what has been built; (2) disqualify restrictions, which check the allowability of a construction before it is built; (3) pruning restrictions, which are actually allowed to *prune* grammar options, based on context; and (4) selectional restrictions, which check the validity of co-occurrence relations on noun phrases and clauses.

A (simplified) partial grammar is given below in Figure 3. It contains rules for assertion, various object options, and several noun phrase options. The interpretation of the assertion rule is that `assertion` constructs a subject, followed by a tensed verb (`ltvr`), followed by execution of the `w_agree` well-formedness restriction, followed by construction of the object. We describe the execution here in sequential terms, because the well-formedness restriction `w_agree` expects that the subtrees corresponding to `subject` and `ltvr` (tensed verb) will have been completed prior to execution of this rule. As we will see in later sections of this paper, we are able to relax this constraint provided that we allow for the ability to suspend and resume execution of restrictions (see section 6).

```
assertion ::=  subject, ltvr, {w_agree}, object.
                          % w_agree checks subject-verb number agreement
object    ::=  vo:        % infinitive (v) + object: eat fish
               veno:      % past participle (ven) + object: eaten fish
               tovo:      % to + verb + object: to eat fish
               ntovo:     % noun + to + verb + object: them to eat fish
               objbe:     % object of be-verb (adjective or prep. phrase)
               assertion; % assertion: they eat fish
               ...
               np.        % direct object

vo        ::=  lvr, object.
tovo      ::=  [to], vo.
ntovo     ::=  {d_to_ahead}, np, [to], vo.
                          % d_to_ahead checks that there is a literal "to"
                          % in the remaining word stream, before building ntovo

np        ::=  lnr;       % noun + left and right adjuncts
               lpror.     % pronoun + left and right adjuncts
lnr       ::=  ln, *n, {w_np_agree}, rn.
lpror     ::=  lpro, *pro, {w_case}, rpro.
```

Figure 3: A Sample Restriction Grammar

The above sample grammar contains several restrictions. The *w_agree* restriction executes on completion of the verb node, to check subject-verb agreement. The w_np_agree restriction checks number agreement between the noun and its left modifiers (e.g., *those chairs*, but not *those chair*). The w_case restriction checks the case of the pronoun: it is nominative if the pronoun is used as the subject, but accusative otherwise (*I find they have been reliable* vs. *I find them to have been reliable*). Pruning restrictions are discussed later, as part of the discussion of the dynamic translator (see section 6.1).

2.5. Grammar Coverage in PUNDIT's Restriction Grammar

PUNDIT has a broad coverage grammar of English. This includes treatment of assertions, questions (yes-no, direct and indirect), and imperative constructions. Our current grammar identifies some forty verb complement types and also handles complex adjectival and noun complements. The treatment of noun phrases includes coverage of pre-nominal and post-nominal constructions, e.g., numerical expressions such as *a two year old child*, *the number two turbine* . Sentence adjunct constructions include adverbs, prepositional phrases, subordinate clauses, and prepositionless time expressions (*I did it this morning*). Also the grammar includes a comprehensive treatment of conjoined structures, as well as a treatment of wh-expressions (see section 4.2).

Because much of our work has been focused on message traffic, PUNDIT also supports a comprehensive, elegant treatment of fragmentary sentences that are characteristic of message text [Linebarger1988]. There are five basic fragment types, including fragments for missing subject (`tvo`: *was repaired*), missing verb (`zero_copula`: *disk bad; disk repaired*), missing subject and verb (`predicate`: *broken since yesterday*), missing object (*engineer repaired*), and noun phrase fragment (`nstg_frag`: *bad drive*). Interestingly, most of these constructions occur as object options in regular English, e.g., *I thought the disk defective.*

Parsing fragmentary sentences adds greatly to the parse search space, since fragmentary structures are highly degenerate and add considerable ambiguity to the grammar. To adapt to this, we have implemented a sequential or (`xor`) construct, that allows us to try for all full sentence analyses before looking for fragment parses. If there are full sentence analyses, then the fragment options are not tried. The following clause describes the behavior of `xor`, although it is not implemented in this way for obvious efficiency reasons.

```
A xor B :-  ( \+ A -> B ; A).
```

Processing run-on sentences (sentences missing a period) is also necessary for handling certain classes of messages. This poses a serious problem of computational efficiency: although we have experimented with the definition a run-on sentence type in the grammar using the `xor` connective, this is enormously costly in terms of performance, and we have found it impossible to obtain reasonable results with a top-down parser. This is one of the problems that has driven us to examine alternative parsing strategies, such as the use of well-formed substring tables. We discuss this issue further in section 6.

3. Restriction Grammar as a Logic Grammar

Restriction Grammar belongs to the general class of logic grammars, the best known member of which is Definite Clause Grammar (DCG) [Pereira1980, Pereira1987]. This class of grammars also includes gapping grammars [Dahl1984], Modular Logic Grammars [Dahl1983] and Definite Clause Translation Grammar [Abramson1984]. Many of these styles of logic grammar have influenced each other, so that they now share certain features that were lacking in Definite Clause Grammar. Restriction Grammar is distinguished from DCGs by several features. RG automates the construction of the (surface) parse tree; the parse tree reflects the set of BNF definitions applied to date, and the construction of the parse tree is done via a pair of parameters inserted when the BNF definitions are converted into Horn Clauses. RG shares with DCGs the notion of restrictions interspersed with rewrite-rules. However, RG allows *no* parameters in the rules. All propagation of information is done via the (partial) parse tree: a restriction traverses the parse tree, locating various constituents (e.g., head of a construction, left-adjunct, subject), and testing various properties of the words associated with these nodes. Finally, the power

of restrictions is limited (in a way that procedure calls in DCGs are not): restrictions can only accept or reject parse trees, they do not build structure. Also, restrictions are written in terms of a limited number of restriction language operators, which constrain the power of restrictions. This contrasts with the ability to insert arbitrary Prolog code into DCGs.

3.1. Implementation of Restrictions

Since restrictions check the well-formedness of the parse tree, they must be able to traverse this tree freely. Each restriction is "housed" at a node (the left hand side of the BNF definition in which it occurs). Its execution assumes that it is "located" at this node for purposes of traversing the parse tree.

For example, the pronoun case agreement restriction `w_case` is executed upon completing a pronoun.

```
lpror     ::=  lpro, *pro, {w_case}, rpro.
```

To determine case, it must look up the tree, starting at the node `lpror`, to find the context of the pronoun: whether it is under a subject node, or under a (complex) object node in the tree. For example, a parse tree of the sentence *I find them to have been reliable* is shown in Figure 4 below. In this sentence, the pronoun *them* occurs under the `np` node of the object node `ntovo`, and thus must be in the accusative case.

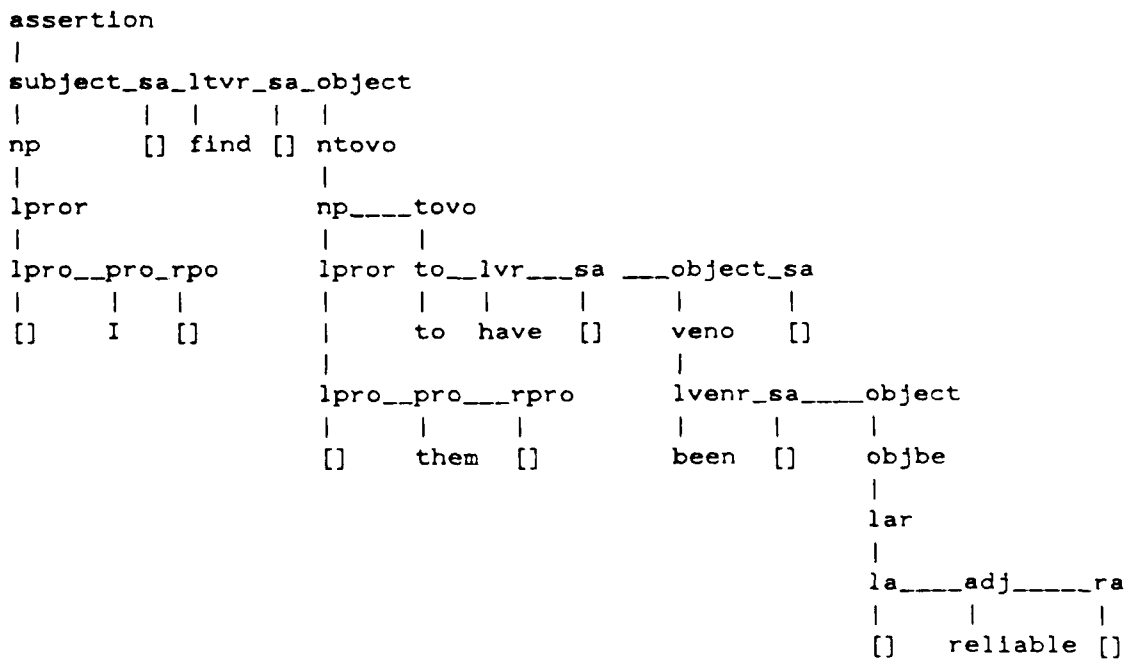


Figure 4: A Parse Tree for *I find them to have been reliable.*

To support free tree traversal, RG uses two recursive data structures, one to maintain forward links, and one to maintain backward links. The first data structure is the tree term (tt). This has fields for the label (name) of the node, first child, next right sibling, word attached, if any, and regularized form (ISR). This is a recursive data structure, where both FirstChild and RightSib are themselves tree terms:

```
tt(Label, FirstChildTreeTerm, RightSibTreeTerm, Word, ISR).
```

To support movement up and to the left in the tree, the path back to the root is passed around as the tree is built during parsing. This too is done via a recursive data structure whose functor gives the direction (up or left), with two arguments: the tree term of the parent or left sibling, and the remaining path back to the root. This data structure is combined with the tt structure in a link data structure as follows:

```
  link(CurrentNodeTreeTerm,PathToRoot)
where
  PathToRoot = up(ParentTreeTerm,PathFromParent)
or
  PathToRoot = left(LeftSibTreeTerm,PathFromLeftSib).
```

For each operation (up, down, left, right), a new link structure is constructed. For the left and down operations, this involves extracting the appropriate field from the tree term and incrementing the path to the root:

```
down(link(StartNode,Path), link(Child, up(StartNode,Path))) :-
   StartNode = tt(_Label,Child,_RightSib,_Word,_ISR).

right( link(StartNode,Path), link(RightSib, left(StartNode,Path))) :-
   StartNode = tt(_Label,_Child,RightSib,_Word,_ISR).
```

To move to the left or up, the correct node is accessed via the path argument (second argument) of link, and the path is appropriately shortened:

```
left(link(_StartNode, left(LeftSib,LeftSibPath)), link(LeftSib,LeftSibPath)).
up(link(_StartNode, up(Parent, ParentPath)), link(Parent,ParentPath).
up(link(_StartNode, left(LeftSib,LeftSibPath),Parent) :-
   up(link(LeftSib,LeftSibPath),Parent).
```

3.2. The Restriction Language

Restrictions are written using a layered approach that makes the tree operations independent of the particular implementation of tree structure. This approach has proved extremely useful in insulating the grammar from changes in the underlying execution mechanism. The lowest layer of operators consist of the primitive tree relation operators described above: up, down, left and right, plus label, which extracts the node label, word which extracts the word attached to a node, and operators (attrb) for accessing lexical subclasses of the word. On top of this layer are a set of *restriction operators* that support various extended tree traversal operations (e.g., iterative ascent and descent),

the `test` operator, which examines a node to determine its type (label), as well as operators to examine the word input stream for optimization purposes. The next layer of routines captures syntactic relations such as *head* (`core`)of a construction, the *main verb*, or the *left/right adjunct* of a construction. Finally, restrictions (e.g., subject-verb or case agreement) are built out of the routines and the restriction operators.

Figure 5 shows a simplified version of the `w_case` restriction. Note that restriction calls appear in the BNF definitions without parameters (see Figure 3). Parameters for current tree location (a `link` term) and for the remaining word stream are added during execution. Thus the procedure definition appears with these two parameters. The *w_case* restriction calls the `core` routine, which accesses the head of a construction (the `pro` in the `lpror` construction).

```
w_case(LPROR,_):-
   core(LPROR,Pronoun),
   up(LPROR,NP),up(NP,Parent),
   (attrb([nominative],Pronoun)->nom_check(Parent);
        (attrb([accusative],Pronoun)->acc_check(Parent);
          true)).

nom_check(Parent):-
   test(subject,Parent).

acc_check(Parent):-
   test(object,Parent);
   test(ntovo, Parent).
```

Figure 5: The Restriction *w_case* for Pronoun Case

4. Meta-Rules in Restriction Grammar

A logic grammar framework lends itself readily to the incorporation of a meta-rule component. Restriction Grammar relies on two kinds of meta-rules, one to handle conjunction and the other for wh-constructions. The meta-rules for conjunction operate on the rules for unconjoined structures, to produce rules which capture conjoinings. For wh-constructions, the BNF definition read-in procedure adds to each definition a set of paired parameters to keep track of gap status: whether a gap is needed and whether it has been found.

The advantage to using meta-rules is three-fold. First, meta-rules can capture the underlying generality of constructions such as conjunction or wh-constructions. It may be possible to enumerate all possible permutations of conjoining (or wh-constructions). But such an approach misses the linguistic generality of the constructions. In addition, the enumeration approach makes for a very unwieldy and unmaintainable grammar. Description of these phenomena via meta-rule preserves the compactness of the grammar, which in turn facilitates maintenance. Finally, meta-rule application is done at "compile" time,

not run time. This means that meta-rules are more efficient than various alternative approaches, such as the standard interrupt-driven approach to conjunction [Raze1976, Woods1973].

4.1. The Conjunction Meta-Rule Mechanism

The conjunction meta-rule mechanism operates on the set of BNF definitions (without conjunction) and produces a new set of grammar rules which cover most cases of conjoining and gapping under conjunction [Hirschman1986]. The meta-rule expands each node of type string or `lxr` to include, as one option, a conjunction followed by a recursive call to the rule. Thus, the (simplified) expansion for `lnr` is:

```
(lnr ::= ln, nvar, rn) =>
      (lnr ::=    ln, nvar, rn;
                  ln, nvar, rn, conj_wd, lnr).
```

This rule states that the `lnr` node can either be expanded as usual, or, via the second option, the usual expansion can be followed by a conjunction word plus a recursive call to `lnr`. (In actuality, the rule is written more efficiently, so that the `ln+nvar+rn` does not have to be rebuilt if there is a conjunction.) Thus BNF definitions can be written without worrying about conjunction. The meta-rule component is then applied to generate automatically the correct rules to support optional conjoining. The notion of such a meta-rule has been elegantly generalized in work by Abramson [Abramson1988].

The conjunction meta-rule component generates rules to handle conjunction only at `lxr` and string type nodes, to eliminate some of the spurious ambiguity that can be associated with treatments of conjunction. The current mechanism handles a variety of conjunctions (*and, or, but*), paired constructions (*both...and, neither...nor*) and "comma-conjunction" (use of comma to take the place of an explicit conjunction in a list such as *apples, oranges and pears*). Since the meta-rule generates a recursive definition, an arbitrarily long series of conjunctions can be handled.

The creation of arbitrary recursive conjoined structures via meta-rule is only half of the solution to conjunction. It is also necessary to account for gapping under conjunction. For example, one can have the following possible conjoinings shown in Figure 6. The issue is how to handle these gaps without creating an enormous proliferation of rules. Our solution is to generate the full `lxr` or string structure, but to allow certain elements to be realized as null elements (`nullc` = null under conjunction). The structure of *the dogs and cats in the yard* is shown in Figure 7. This is similar to a solution described by Sedogbo, who did not, however, use meta-rules to generate his conjoined structures [Sedogbo1984].

Here the `nullc` elements are place holders for the gapped elements, indicated by arrows. It is necessary to distinguish the gaps from "real" words, but it is also necessary to know how to fill them in, for purposes of checking

```
1) No gap, adverb has local scope:
    The dog slept and the cat played in the yard.
2) Gapped adverbial (adverb has scope over conjoined expression)
    The dog slept (in the yard) and the cat played in the yard.
3) Gapped object:
    The dog chased (the bird) and the cat ate the bird.
4) Gapped verb:
    The dog chased the squirrel and the cat (chased) the bird
5) Gapped subject:
    The dog chased and (the dog) caught the squirrel.
```

Figure 6: Gapping in Assertion-Level Conjunction

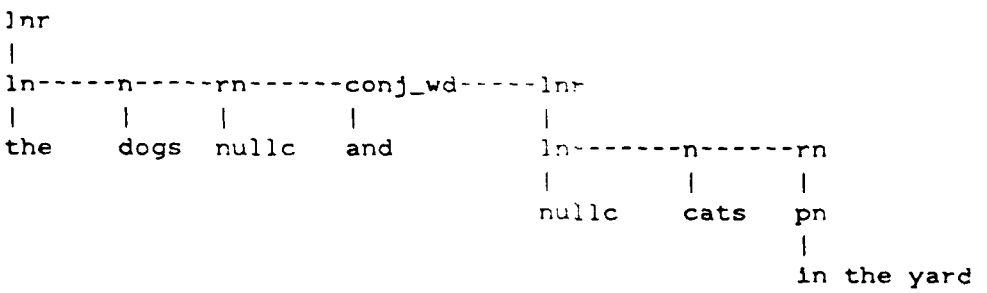


Figure 7: Parse with Gapped Elements under Conjunction

selection and other restrictions. This is accomplished by setting pointers from the gaps (`nullc` elements) to their fillers. The existence of the `nullc` node distinguishes them as gaps; the pointer enables restrictions to find the filler for the gapped element.

This treatment of gaps as null elements provides a highly regular structure. However, a problem remains, namely to control the use of gaps (`nullc` elements), in order to prevent spurious ambiguity and serious overgeneration in the grammar. Gaps are controlled by the use of restrictions, which allow gaps only in certain places. For example, an object gap is allowed only if the next word is a conjunction. A verb gap is allowed only if there is no subject gap, etc. These restrictions are inserted during application of the meta-rule. Slightly different rules apply to string constructions vs. `lxr` nodes, therefore these are handled by two distinct meta-rules.

Figure 8 shows a more complete version of the conjunction rule, for string nodes. Notice that a number of restrictions are inserted, in part to control for `nullc` placement.

The rule in Figure 8 covers two cases: one for *either...or* and and second for the rule, followed by an *optional* conjoined element. The restrictions {wconj_comma}, {wconj_nullcVerb}, {wnullcObj} and {d_check_nullc} have all been inserted. These constrain the comma conjunction, verb as `nullc` and object as `nullc` respectively.

```
(STRING ::= Rule -> Isr) =>
    (STRING ::=
% Either-or rule
      [either], Rule,
        [or], sa, STRING, {wconj_comma}, {wconj_nullcVerb}, {wnullcObj}
                          -> [or,Isr,STRING]),
    (STRING ::= Rule,
% Option 1: with conjunction
        ((conj_wd, {d_ln_conj}, sa, STRING,
              {wconj_comma}, {wconj_nullcVerb}, {wnullcObj}, {w_ln_conj}
                    -> [conj_wd, Isr, STRING])
% Option 2: no conjunction
      ; {d_check_nullc} -> Isr)).
```

Figure 8: Meta-Rule for String Constructions

One problem in handling gapping is the reconstruction of the gapped information for purposes of selection and semantics. The treatment in RG simply assigns a pointer from the gapped element to the "real" element. This provides for a mechanism to distinguish actually occurring elements from "virtual" elements, but also supports access to the filler of the gap, for checking selection. For semantics, the solution is even simpler. Semantics interfaces to the syntax via the ISR. Note that the rule for the ISR formation (indicated after the -> in the BNF definition) has a conjunction inserted, with the two ISRs of the two conjuncts as its operands. These are simply copied in the meta-rule. The relationship of the ISR computation to conjunction is described further in section 5.

4.2. Wh-constructions

The wh-constructions are also handled by meta-rules. The function of the meta-rule is to introduce parameters into each definition, so that gap information can be passed around, namely the need for a gap, or the fact that a gap has been found. This makes the handling of wh-constructions largely invisible to the grammar writer, who need only worry about routine constructions.

The treatment of wh-constructions covers questions, relative clauses and indirect questions (*I don't know what they want*). It also supports correctly the interaction between conjunction (and its gaps) and the wh-constructions (and their gaps). Wh-constructions present a problem for most styles of grammar because of a long-distance dependency between the wh-word (which signals the creation of a gap) and the gap, found in the following clause, possible deeply embedded, as in *The hat which I thought you knew that I had bought ( ) turned out to be too small.* The most obvious way to handle this is via parameter passing. This is the idea behind the GPSG slash category treatment[Gazdar1981] and also behind the logic grammar treatment in Pereira and Schieber [Pereira1987].

The treatment in Restriction Grammar [Hirschman1986] uses the same idea of parameterization, but in keeping with Restriction Grammar, hides the explicit parameters from the grammar-writer. A set of four annotations flags the gapping status of a rule as follows:

```
<<    Create gap
>>    Realize gap
<>    Disallow gap
><    Copy gap status under conjunction
```

The fronted wh construction *creates the need* for a gap. The gap is realized by allowing the appropriate element (e.g., noun phrase, adverb) to be realized as an empty string, written in string grammar as `nullwh`. Using these annotations, the rules to support a wh-question are as shown in Figure 9.

The gap annotations are converted to a pair of parameters at rule read-in time. The annotation << sets the parameter to `need_gap/Out`, while the annotation >> sets the parameter to `need_gap/found_gap`. Thus a gap can *only* be realized if the need for a gap has been created, and the gap *must* be realized before exiting the rule marked >>. This captures the notion of the scope of the gap very simply, especially when coupled with the *no-gap* marker <>. The no-gap marker does not allow the status of the gap to change: it sets the parameters so that the input parameter = the output parameter. Note that most definitions remain unchanged: an unmarked definition will simply pass along the gap status of its parent to its first child, and pass back the gap status of the last child on up the tree, to its next sibling or back to the parent.

The flow of information through the parse tree can be seen in Figure 10 (sentence adjunct slots are omitted and several rules are simplified, for reasons

```
wh_question ::=
     sa, <<whQ, >>yesnoq.

<<whQ ::= [how];   % adverbial gap
        [who];   % noun gap
        ...
        [whom].  % noun or adjective gap

yesnoq ::=
    sa, ltvr, sa, subject, {wagree},{w_sai}, sa, object, sa.

subject ::= ...
object  ::= ...

np ::= lnr; lpror.

>> np ::= nullwh.
```

Figure 9: Rules for Wh-Question

of space).

One of the elegant features of this treatment is the ease of integration into the treatment with conjunction. The basic insight is that when the conjunction meta-rule is applied, the gap-status of the conjuncts should be identical: if one conjunct realizes a gap, the other must too; if one does not, the other cannot. This is the function of the fourth annotation, ><. When "translated" at rule read-in time, this annotation causes the parameters of the two conjuncts to be made identical.

A detailed treatment of wh-constructions requires a number of refinements to the basic schema outlined above. In particular, it is necessary to pass along some additional information, embedded in the parameters, about type of gap (noun phrase, adjective, adverb). In addition, some extra information is required to construct the appropriate ISR. However, one of the advantages of this treatment is that almost no restrictions are required. This is because the parameters carry the information forward to the appropriate point. It is of course possible to implement a treatment of wh-phenomena without using parameters. However, such a treatment involves a great deal of search up and down the tree, checking that if there is a wh-word, there is exactly one gap in its scope, and also the converse: each gap is in the scope of exactly one wh-word. The parameter-based treatment is far simpler and far more efficient. By doing the parameterization at rule read-in time, it is possible to insulate the grammar-writer from the need to parameterize rules by hand.

5. Intermediate Syntactic Representation

As described in the previous section, the Restriction Grammar used in PUNDIT is a large and complex broad-coverage grammar of English. While the level of complexity of the surface structure parse tree is necessary for doing correct syntax, not all of that complexity is necessary f ther semantic and pragmatic processing. In order to buffer PUNDIT . . antic and pragmatic

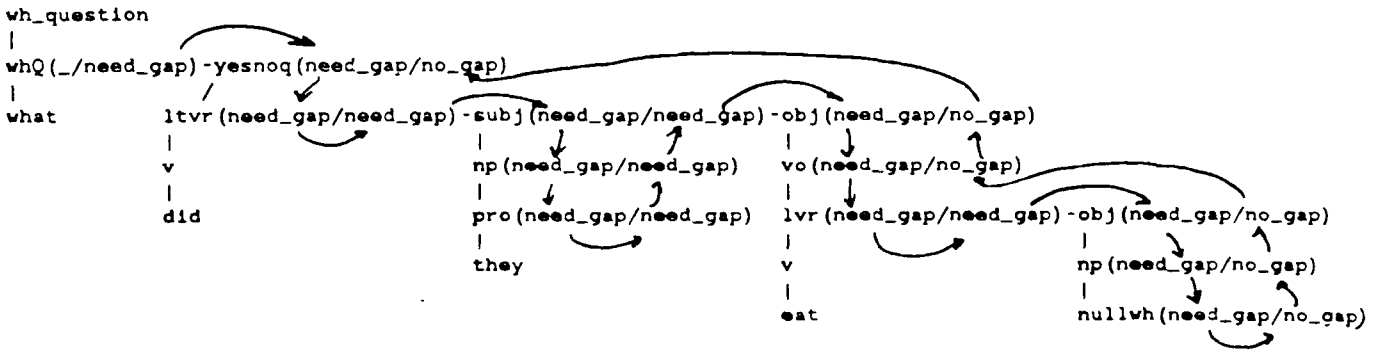


Figure 10: Flow of Gap Information in *What did they eat?*.

components from the complexity of the surface structure parse tree, we developed the *Intermediate Syntactic Representation* (ISR), which provides a more canonical representation of the information expressed in the parse tree. The ISR is an operator/operand form that regularizes the predicate-argument-modifier relations implicit in the parse tree. In addition, the ISR provides a clean separation between syntax and semantics, allowing changes in the grammar to be made independently of semantic changes.

An example of an ISR expression is given below for the sentence *I found they have been reliable.*

```
[past,
   find,
      subj([pro([i,singular,A])]),
      obj([present,
             perf,
                be,
                   subj([pro([they,plural,B])]),
                   adj([reliable])])]
```

The ISR is represented as a Prolog term. The top-level operator of this ISR is the tense marker 'past' which is an operator that takes one argument (an OP1). Using Prolog list syntax, if the first element of the list is an OP1, then its argument is the remainder of the list. All tense and aspectual markers are OP1's. The first element of the remainder of the list is the verb *find*. Since verbs have variable numbers of arguments plus optional modifiers, the remaining elements of the list are taken to be the arguments of the verb, followed by its modifiers, if any. Here, the subject and object are verb arguments. The subject is the singular pronoun `I`, and the object is itself a clause, with OP1's `present` and `perf` indicating tense present and aspect perfect, and verb `be`. This ISR can be compared with a similar one for the sentence *I found them to have been reliable.*

```
[past,
   find,
      subj([pro([i,singular,A])]),
      obj([untensed,
             perf,
                be,
                   subj([pro([they,plural,B])]),
                   adj([reliable])])]
```

Here the differences in syntactic structure which are not relevant to semantic and pragmatic processing have been removed. The only difference in the ISR of these two sentences is in the tense marker of the object clause.

In order to construct the ISR, every rule in the grammar is annotated with an expression in the *Translation Rule Language* (TRL). These TRL expressions declaratively indicate how the ISR of a node should be constructed from the ISR of its children. The ISR's of terminal nodes are derived from the lexical entries of the words attached. Thus, the construction of the ISR is strictly

compositional.

Evaluation of the ISR goes on in two stages. In the first stage, TRL expressions are evaluated, which gives rise to a list structure representation for each node. These structures, however, may contain lambda expressions whose values may not be available at the current node. Thus, a second stage *simplification* is needed in order to perform lambda application and to remove unnecessary list structure.

TRL expressions are associated with grammar rules using a Prolog infix operator `->` which has the body of a BNF expression on its left, and a TRL expression on its right. For a simple example, consider the grammar rule for `object` when the object of the verb is a simple noun phrase.

```
object ::= np
         -> lambda(Verb,lambda(Subject,[Verb,subj(Subject),np])).
```

The TRL expression here is a function of two arguments (actually, two functions of one argument each). The body of the lambda expression constructs a predication whose operator is the verb, with the subject and object as operands. The `subj` label wrapped around the subject inserts a syntactic role marker in the ISR. These markers are called 'semlabels' (semantic labels) because they are used in semantic processing to map syntactic constituents to semantic roles. The `np` atom in the TRL expression is a command to insert the ISR of the np child into the ISR of the object node at this point. When the object rule is completed, its ISR will not be simplifiable because the arguments to the function will not be available. One parent node of object is `assertion`, whose (simplified) grammar rule is:

```
assertion ::= subject, ltvr, object -> [object, ltvr, subject].
```

For the sentence *I found them*, evaluating the TRL expressions for assertion, subject, verb, and object yields the structure:

```
[lambda(Verb,
    lambda(Subject,
       [Verb,
          subj(Subject),
          obj([pro([they,plural,B])])])),
 find,
 [pro([i,singular,A])]]
```

The use of the list notation in the ISR is overloaded, indicating both operator/operand structure and lambda application. In this case, because the first element of the list is a lambda expression, we know that this is an instance of lambda application. The ISR of the object, a function of verb and subject, is applied to the ISR's of verb and subject, to produce an operator-operand representation of the clause:

```
[find,
    subj([pro([i,singular,A])]),
    obj([pro([they,plural,B])])]
```

The declarative nature of the Translation Rule Language allows it to integrate smoothly with the meta-rule component. The meta-rule for conjunction derives the TRL expression for the new rule from the TRL expression of the original. Every lxr or string rule will have a TRL expression of the form:

```
nonterminal ::= ... -> TRL.
```

The meta-rule for conjunction will create a new rule of the form:

```
nonterminal ::= <rule body> -> TRL
      ; <rule body> conj_wd nonterminal -> [and, TRL, nonterminal].
```

This expression uses the OP2 *and*, which is an operator that takes exactly two arguments. This straightforward transformation on the TRL will generate the correct TRL for any conjoined lxr or string nodes.

Because the ISR is constructed using a variant of the lambda calculus, there is substantial flexibility in the methods used to evaluate it. For efficiency, we have chosen to evaluate the ISR lazily. Currently in PUNDIT, semantics is only done at the end of noun phrases and clauses. Therefore, the value of the ISR is only needed at the completion of parsing of noun phrase and clause nodes. The ISR evaluator takes advantage of this by not constructing the ISR expression for a node until semantics asks for its value. For nodes other than noun phrases and clauses, the ISR is constructed when it is needed to construct the ISR of its parent. Using this approach, ISRs of nodes that are constructed but that fail prior to being incorporated into a noun phrase or clause do not ever have ISRs constructed for them. With PUNDIT's current strategy for computing semantics, this gives us considerable efficiency gains. We have also considered using an eager evaluation style. This would be more appropriate if semantics were being done on more partial constituents, or if semantics could be used to predict syntactic structure. Due to the declarative nature of the Translation Rule Language, changing from a lazy to an eager evaluation scheme would require no changes to the grammar.

6. Parsing with Restriction Grammar

During the past 8 years we have constructed many different parsing mechanisms for Restriction Grammars. The first parser was a top-down left-to-right backtracking interpreter. This was followed by a translator, then the *Dynamic Translator* [Dowding1987], which will be described shortly. More recently, we have concentrated on bottom-up and well-formed substring table parsers for Restriction Grammars.

6.1. Top-down Parsing

A (top-down) interpreter parses a string as a phrase of a given category by choosing a grammar rule of that category, dividing the phrase into sub-phrases, and parsing those sub-phrases into the designated categories. At run-time the interpreter requires as input not only the string that is to be parsed, but also the set of grammar rules needed to parse it. Alternatively,

the process can be broken down into two phases, the translation phase and the run-time phase. The translator takes the complete set of grammar rules and produces a set of Prolog procedures (Horn Clauses) which, when called at run time, will parse the phrase in exactly the same way that the original grammar would have. The translation phase converts the information explicit in the grammar rules into information implicit in the Prolog procedures. The translation phase requires the set of grammar rules but does not have the input string available to it, while the run-time phase after translation has access to the input string, but does not have explicit access to the original grammar rules.

In contrast to these traditional approaches, PUNDIT uses a single mechanism, the Dynamic Translator, that takes advantage of the strengths of translation and interpretation without the corresponding disadvantages. The Dynamic Translator has available to it both the input string and the grammar rules (like an interpreter), but also makes use of both a translation and a run time phase (like a translator). In the Dynamic Translator, the translated code runs in co-operation with an interpreter to parse a sentence. Although one might thus expect that the speed of the Dynamic Translator to be intermediate between an interpreter and a translator, the Dynamic Translator is substantially faster than either. This added efficiency is gained by the use of the Dynamic Rule Pruning mechanism, an inherently interpretive device that dynamically prunes the search space.

The Dynamic Rule Pruning mechanism uses information available only at run-time to reduce the number of options that must be considered for certain grammar rules. This information includes, for example, both the input word stream and the partially constructed parse tree. Since the rule pruning mechanism is a source-to-source transformation (considering the grammar rule bodies as source language expressions) that can only be applied at run-time, when an invocation of the rule pruning mechanism is translated, a call to the interpreter is generated that will interpret the source code generated. When the interpreter attempts to evaluate a nonterminal name, control will revert to the translated code. Thus, the Dynamic Translator allows interleaved execution of translated and interpreted code. Reducing the number of options using Dynamic Rule Pruning and the Dynamic Translator eliminates extraneous paths from the search space and greatly increases the efficiency of the parsing process (measured as a factor of 20 in parsing the sentences from part of our CASREP corpus) [Dowding1987].

6.2. Bottom-up and Well-formed Substring Table Parsing

The previous section describes the traditional top-down left-to-right backtracking parsers for Restriction Grammars. While these parsers can be made to run acceptably fast, they suffer from a number of criticisms, including that their worst case performance grows exponentially with sentence length and they

cannot handle left-recursive grammar rules. The obvious answer to these criticisms is to use a bottom-up or well-formed substring table parser. However, previous attempts at building bottom-up parsers for string grammars have resulted in systems that were extremely slow and required substantial modifications to the grammar. The primary reason for this inefficiency is the restriction handling mechanism that these parsers employ.

One of the two inputs to a restriction is the partial parse tree. With a top-down parser, the task of constructing the partial parse tree is straightforward. With a bottom-up or WFST parser, this is much more difficult. The partial parse tree will contain a combination of completed and active nodes [Winograd1983]. A completed node is one that has all of its children fully constructed. In contrast, an active node is one where at least one of its children has not been constructed. Obviously, all the nodes beneath a completed node are themselves completed, and all the nodes above an active node are themselves active. The difficulty for constructing the partial parse tree for a bottom-up or WFST parser lies in determining the active nodes above the node that you are currently constructing.

To get around this problem, previous implementors of bottom-up parsers for string grammars have decreed that restrictions can only look down. While this constraint allows bottom-up parsers to be built, the resulting systems suffer from a number of severe criticisms, the worst being that existing string grammars cannot be executed on these parsers without substantial modification. In RG, restrictions are "housed" at a specific node. This node is the starting point for the restriction's tree traversal; it also controls when the restriction is executed. Basically, in a bottom-up parser, any restriction that travels up the parse tree will have to be rehoused in the grammar at the highest possible point that it could have traveled to, and the restriction itself will have to be rewritten. In addition to this problem, housing restrictions high in the parse tree has severe negative effects on efficiency. During parsing, besides checking for the satisfaction of some linguistic constraints on the applicability of the context-free phrase structure rules, restrictions have the beneficial effect of ruling out ungrammatical parse paths. Clearly, to gain the most efficiency a restriction should execute as early in the life of an ungrammatical parse path as possible. Housing restrictions high in the parse tree has the effect of not executing them until very late in the life of the ungrammatical path. To make matters worse, restrictions must be housed as high as they might climb in the worst-possible case, even if in most cases they don't need to climb upwards at all.

Although restrictions resemble embedded Prolog goals used in Definite Clause Grammars, they are actually much more constrained. Restrictions must be written in the Restriction Language, so they will not have any side-effects, will not construct any Prolog terms, and will not unify any variables that existed prior to the restriction call. These constraints lead to an important property of restrictions, namely that they can be executed in any order. We have

made use of this property in developing a continuation-based restriction interpreter [Dowding1989]. This is an interpreter for the Restriction Language which monitors restriction execution for an attempt to climb up to an active node. If this is attempted, then the interpreter will halt the execution of the restriction, and return a continuation, along with a logic variable that represents the active node that the restriction tried to climb to. The bottom-up parser monitors this variable until it becomes instantiated, at which point it resumes restriction execution. For the WFST parser, execution of a restriction is resumed when a node with associated continuations is attached to a parse tree.

We have used this continuation-based interpreter to build both bottom-up and WFST parsers Restriction Grammars. These parsers are able to execute existing Restriction Grammars without modifications. They have the ability to execute many restrictions as early in the parse as a top-down parser would, but without the exponential growth in parse times typical of top-down parsers. This work is still in an experimental stage, but we believe that it offers great potential in handling phenomena such as run-on sentences and sentences for which we can not obtain a complete parse.

7. Parallelism in RG

Parsing is a search problem -- there is a great deal of non-determinism in parsing, especially given a large grammar. This stems from two sources: first, the need to explore paths that end in failure. For example, given the initial string *the bus stops*, we have no idea if we have a noun phrase (*the bus stops are indicated in red*) or subject + verb (*the bus stops here*). Second, there may be ambiguity, particularly in the absence of domain-specific knowledge, which results in multiple parses for a given sentence, e.g., *starting air compressor failed = starting the air compressor failed* or *the starting air compressor failed.* The correct parse depends on domain information. The search space explosion in a large grammar, particularly when dealing with fragmentary input or long sentences, makes parsing an ideal application for exploiting or-parallel search.

Over the past two years, we have conducted a set of experiments that indicate that substantial speed-ups can be obtained by exploiting or-parallelism at the level of grammar rule disjunctions in RG. Our evidence comes from two sources. First, we ran a series of simulation experiments assuming a shared-memory multi-processor architecture running an or-parallel Prolog. Second, we have been able to run a preliminary series of experiments on an actual or-parallel Prolog implementation, which confirmed the availability of useful or-parallelism during parsing.

In the simulation experiments, we searched for all solutions (parses) in parallel. One of the attractive features of this paradigm is that the search paths can be pursued independently of one another: when a grammar rule has a disjunction, the processing simply splits: one processor pursues the parse, applying the first disjunct, and runs until completion or failure; the other

processor does likewise, except that it must copy the current "state of the parse", naming all logic variables apart, before initiating execution of the second disjunct. On failure, a processor simply returns to the processor pool, and is assigned a new parse state and rule disjunction to pursue.

This simple approach to spawning or-parallel processes turned out to be sufficient to obtain expected speed-ups of 20-30 fold, running to all parses for a set of sentences from Navy messages [Hirschman1988]. Our initial concern was that the copying costs to copy a given "parse state" might be prohibitive. Experimentation showed that the key factor in generating useful or-parallelism was the ratio between process granularity (the median process duration between process spawns) to the time for process start-up (including the time to copy the parse state). We can rephrase this as how much time is spent spent doing useful work in parallel vs. the time spent in overhead, setting up the parallel processes. It turns out that if this ratio goes below 2:1, more than half the available speed-up is lost.

RG has several properties that made it possible to exploit the parallelism effectively. First, the parse state to be copied turns out to be the *path* from the node under construction to the root. This is where the tree is "growing", that is, where variables will be instantiated as the parse continues. Second, because of restrictions and the automatic tree building, the granularity of the processes turned out to be big enough relative to the estimated copying cost: our simulation results showed an average process duration of 9 ms., vs. an estimated copying time of 3 ms., to give a 3:1 ratio, enough to make effective use of the available parallelism.

These results were confirmed by runs on the Aurora Prolog system, the or-parallel Prolog system under development at the Swedish Institute for Computer Science (SICS). We achieved average speed-ups of greater than 10-fold using only 12 processors. For several of the tested sentences, these results exceeded the maximum predicted speed-ups given by the simulation experiments. Furthermore, we have reason to be very optimistic that the performance will become even better. We measured the speed-ups using 4, 6, 8, 10, and 12 processors, and there did not seem to be any reduction in the speed-up improvement as more processors were added, indicating that we had not yet exhausted the available parallelism.

We are now in the process of planning a series of further experiments, both to relate these results to our earlier simulation experiments, and to increase the number of processors.

8. Future Directions

We are actively investigating a number of techniques designed to increase the speed and flexibility of Restriction Grammar. Much of this work is being done in the context of a new research direction on Spoken Language Understanding. Our plan is to couple speech recognition technology with the

PUNDIT system. Spoken Language Understanding (SLU) imposes substantially increased computational demands on the language understanding subsystem, due to the indeterminacy at the level of the input signal. While speech recognition has made major advances in recognizing continuous, speaker-independent input, the best research systems still range between 80-95% word accuracy for medium-sized (1000 word) vocabularies. Moreover, these word accuracy figures assume that the language obeys a highly constrained "bi-gram" model of what words can follow a given word. Without the constraint of the bi-gram model, word recognition accuracy drops to about 60%. Given an 8 word sentence, this translates into a sentence-level accuracy ranging from 5% (without bi-gram model) to 74% (with bi-gram model).

This places an enormous burden on the language processing subsystem: if it can supply tight enough constraints, it can push sentence recognition accuracy up reasonably high, since it can narrow the search space of the SLU system, to help focus more quickly on meaningful sequences of words. In addition, a linguistically-based grammar can discard some of the strings of words that are accepted by a "dumb" (non-linguistic) model; that is, acoustically, the sequence "show me the ship" and "show meet a ship" may have equally good scores, but only the first is grammatically and semantically meaningful.

Given the massive indeterminacy in translating the acoustic input into a string of words, we believe that the language and speech processing subsystems must be tightly coupled, interacting as e. new word hypothesis is generated, to help to score the resulting word sequences. The SLU system must be able to maintain a (large) number of competing word-sequence hypotheses, each associated with some score (and perhaps, with some partial linguistic analysis). A control mechanism must choose which hypotheses to pursue, for how long, and in what order. In short, the natural language engine must provide not only understanding of the input, but in the process, use this understanding to provide tight filtering of candidate word sequences.

To support this type of search, we need to push RG in some of the directions outlined earlier: use of well-formed substring tables (to avoid recomputation); early application of constraints; eager evaluation of the ISR to support selection on partial constituents. In addition, opportunities for parallelism abound in this application. Each hypothesized word sequence represents a traditional natural language understanding problem: assign the word sequence a syntactic and semantic interpretation -- except that in this case, we only know the initial words in the sequence; we do not necessarily know that we are at the end of an utterance (assuming a left-to-right processing paradigm). This immediately gives us two levels of parallelism: parallelism within the analysis of a given word sequence; and exploring multiple word sequences in parallel.

Although we are now asking these questions in light of the extreme demands put on a natural language system by the problem of understanding spoken input, it is clear that this research will benefit written text processing

applications as well. In particular, real messages often contain incomplete or run-on sentences, for which a complete parse is not available. Also, the larger the grammar (and the longer the sentence), the more possible ambiguity can arise, making it important to apply constraints early. The requirements for real-time message understanding are beginning to emerge, and these proposed extensions to Restriction Grammar should have significant pay-off in addressing these needs as well.

9. Acknowledgements

The authors would like to acknowledge the contributions of a number of other researchers. The linguistic coverage of the PUNDIT grammar is due in large part to Marcia Linebarger. The development of the PUNDIT system as a whole is due to the Language Understanding Group at Paoli Research Center, including Catherine Ball, Deborah Dahl, Korrinn Fu, Francois Lang, Marcia Linebarger, Lewis Norton, Martha Palmer, Rebecca Passonneau, and Carl Weir. The parallelism work has been carried out with major contributions from Bill Hopkins and Bob Smith. The results from the Aurora or-parallel Prolog were done working with Andrzej Ciepielewski at the Swedish Institute of Computer Science. Finally, we would like to thank Lewis Norton and Martha Palmer for their helpful comments on a draft of this paper.

REFERENCES

[Abramson1984]
Harvey Abramson, Definite Clause Translation Grammars. In *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, New Jersey, Feb. 6-9, 1984, pp. 233-241.

[Abramson1988]
Harvey Abramson, Metarules and an Approach to Conjunction in Definite Clause Translation Grammars: Some Aspects of Grammatical Metaprogramming. In *Proc. of META'88*, Bristol, UK, June 1988.

[Dahl1986]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[Dahl1987]
Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Passonneau, Nominalizations in PUNDIT, Proceedings of the 25th Annual Meeting of the ACL, Stanford, CA, July, 1987.

[Dahl1983]
V. Dahl and M. McCord, Treating Co-ordination in Logic Grammars. *American Journal of Computational Linguistics* **9, No. 2,** 1983, pp. 69-91.

[Dahl1984]
V. Dahl and H. Abramson, On Gapping Grammars. In *Proc. Second International Conference on Logic Programming,* Uppsala, Sweden, 1984, pp. 77-88.

[Dowding1987]
John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, Presented at the 2nd International Workshop on Natural Language Understanding and Logic Programming, Vancouver, B.C., Canada, 1987.

[Dowding1989]
J. Dowding, A Continuation-Based Interpreter for Restrictions, presented at the Workshop on Natural Language and Logic Programming, Stockholm, April 1989.

[Gazdar1981]
G. Gazdar, Unbounded Dependencies and Co-ordinate Structure. *Linguistic Inquiry* **12**, 1981, pp. 155-184.

[Grishman1973]
R. Grishman, N. Sager, C. Raze, and B. Bookchin, The Linguistic String Parser. *AFIPS Conference Proceedings* **43**, AFIPS Press , 1973, pp. 427-434.

[Harris1962]
Z. Harris, *String Analysis of Sentence Structure*, The Hague, 1962.

[Hirschman1989]
Lynette Hirschman, Martha Palmer, John Dowding, Deborah Dahl, Marcia Linebarger, Rebecca Passonneau, Francois-Michel Lang, Catherine Ball, and Carl Weir, The Pundit Natural Language Processing System. In *Proceedings of the Conference on Artificial Intelligence Systems in Government,* Washington, DC, March, 1989.

[Hirschman1986]
L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* **3**(4), 1986, pp. 299-328.

[Hirschman1988]
L. Hirschman, A Meta-Treatment of Wh-Constructions in English . In *Proc. of META '88,* Bristol, UK, June 1988.

[Hirschman1988.....]
L. Hirschman, W.C. Hopkins, and R.C. Smith, Or-Parallel Speed-Up in Natural Language Processing: A Case Study. In *Proc. of the 1988 International Joint Conference on Logic Programming*, Seattle, WA, August 1988.

[Lang1988]
F. Lang and L. Hirschman, Improved Parsing Through Interactive Acquisition of Selectional Patterns. In *Proc. of the Second Conference on Applied Computational Linguistics*, Austin, TX, February 1988.

[Linebarger1988]
M. Linebarger, D. Dahl, L. Hirschman, and R. Passonneau, Sentence Fragments Regular Structures. In *Proc. of the 1988 Annual Conference on Computational Linguistics*, Buffalo, NY, June 1988, pp. 7-16.

[Palmer1986]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[Passonneau1988]
Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics* **14**(2), June 1988, pp. 44-60.

[Pereira1980]
F. C. N. Pereira and D. H. D. Warren, Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* **13**, 1980, pp. 231-278.

[Pereira1987]
F.C.N. Pereira and S.M. Schieber, . In *Prolog and Natural Language Analysis*, Center for the Study of Language and Information, Stanford, 1987.

[Raze1976]
C. Raze, A Computational Treatment of Coordinate Conjunctions. *American Journal of Computational Linguistics* microfiche **52**, 1976.

[Sager1981]
N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[Sedogbo1984]
C. Sedogbo, A Meta Grammar for Handling Coordination in Logic Grammars. In *Proc. of the Conference on Natural Language Understanding and Logic Programming*, Rennes, France, September, 1984, pp. 137-150.

[Winograd1983]
T. Winograd, *Language as a Cognitive Process, Volume I: Syntax*. Addison-Wesley, Reading, Massachusetts, 1983.

[Woods1973]
W. A. Woods, An Experimental Parsing System for Transition Network Grammars. In *Natural Language Processing*, R. Rustin (ed.), Algorithmics Press, New York, 1973, pp. 145-149.

PORTING PUNDIT TO THE RESOURCE MANAGEMENT DOMAIN

Lynette Hirschman, François-Michel Lang,
John Dowding, Carl Weir
Paoli Research Center
Unisys Defense Systems
P.O. Box 517
Paoli, PA 19301
Arpanet: hirsch@prc.unisys.com

INTRODUCTION

This paper describes our experiences porting the PUNDIT natural language processing system to the Resource Management domain. PUNDIT has previously been applied to a range of messages (see the paper *Analyzing Explicitly Structured Discourse in a Limited Domain: Trouble and Failure Reports* by C. Ball (appearing in this volume), and also [Hirschman1989]. However, it had not not been tested on any significant corpus of queries, such as that represented by the Resource Management corpus. Our goal was to assess PUNDIT's portability, and to determine its coverage of syntax over this domain. Time constraints precluded testing of the semantic component, but we plan to report on this at subsequent meetings. We performed this port with the intention of coupling PUNDIT to the MIT SUMMIT speech recognition system. This work is described in another paper in this volume, *Reducing Search by Partitioning the Word Network*, by J. Dowding.

Our philosophy in porting has been to *tune* the system to a new domain, rather than rewriting the grammar or building the grammar from scratch. The rationale for this approach is to continue to develop the coverage of PUNDIT's grammar; each new application should motivate principled extensions to the system that can also apply to other domains. Thus, over time, the coverage of PUNDIT has grown to cover a very large portion of English, and each succeeding port requires less effort. The disadvantage of this approach is that as the coverage grows, the grammar becomes "looser" -- the number of parses for any given word sequence tends to increase and also the grammar tends to overgenerate, letting through constructions that are not grammatical.

This philosophy is quite different from the "language modeling" approach taken by some groups working in speech recognition. The language modeling approach has as its goal the development of a *minimal* covering grammar needed to describe the phenomena observed in the particular corpus. The benefit of the language modeling approach is that it produces a very tight, highly constrained grammar. The disadvantage is the porting cost, and a very fragile system, whose syntactic boundaries are very easy to exceed.

Our approach to lexicon development has the same focus as our approach to syntactic coverage: to try to capture the general English definitions, rather than to limit ourselves to the particular domain-specific usages encountered in the training data. The rationale is also similar to that used in the syntactic component: generation of lexical entries is a time-consuming process; our goal is to develop a broad coverage system, so when entering a word in the lexicon, we enter the general English categories for the word. In many cases, this provides a much more general definition than

* This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research; and in part by internal Unisys R&D funding.

what is specifically required by an application. For example, the word *alert* occurs exclusively as a noun in the Resource Management domain. However, it must be classified as an adjective and a verb if the entry is made general to English.

The challenge for the broad-coverage grammar/lexicon approach is to develop methods of tuning the grammar and the lexicon to the particular corpus. It is clear that integration of PUNDIT with a speech recognition system will require that we bring to bear as many constraints as possible, in an attempt to prune the explosive search space that results from indeterminacy in analysing the acoustic signal. We discuss several possible approaches to tuning both the grammar and the lexicon in the final section of the paper. What these results provide is a solid indication that our porting strategy is successful: only a very modest effort was required to obtain reasonable results in the Resource Management domain (85% of the training sentences and 76% of the test sentences received a correct parse, given a porting effort of 10 person-weeks). The next steps will be to add semantics and pragmatics, and to develop techniques for (semi-) automatically tuning the grammar to a new domain.

THE PORT

As mentioned above, in this initial experiment, we undertook only the syntactic processing of the Resource Management training and test corpus. In the PUNDIT system, the syntactic stage consists of the generation of a detailed surface parse tree and the construction of a regularized *Intermediate Syntactic Representation* or ISR. The ISR uses an operator/argument notation to represent the regularized syntax. The regularization includes insertion of omitted constituents in relative clause constructions or as a result of various raising and equi operations. In addition, we performed some limited experiments running with selection, with provides a shallow (selection-based) semantic filtering during parsing [Lang1988].

The tasks associated with the port are summarized below, with estimates of the time in person-weeks (PW). The total elapsed time was 1.5 months; the total port time was 10 person-weeks.

Steps in Porting PUNDIT
(4 PW) 1. Build lexicon
(3 PW) 2. Run training sentences
(1 PW) 3. Build Knowledge Base
(2 PW) 4. Collect selection patterns

THE LEXICON

The final lexicon consisted of approximately 1100 words; this number is greater than the usually quoted vocabulary size for the resource management corpus, due to the inclusion of a number of multi-word expressions in our lexicon, particularly for handling geographic names (*Bering Straights*, *Gulf of Tonkin*). Of these, approximately 450 words were already in our general lexicon (which is still quite small, some 5000 words). We entered the remaining 650 words. This total number represents a mix of general English entries (some 150 words), ship names (200), numbers (about 50, handled by the *shapes* component for productive expressions), place names (150), and some domain-specific entries (approximately 100), which were kept separate from the general English lexicon (e.g., *hfdf*).

SYNTAX

Changes to the syntax focused on adding coverage, but not removing any definitions. It even turned out that our treatment of fragmentary or incomplete sentences [Linebarger1988] was needed to run the resource management corpus, for sentences such as *The Kirk's distance from Hornet*. A few months prior to the beginning of the Resource Management port, we had added a comprehensive treatment of wh-expressions [Hirschman1988], which includes both relative clauses and question forms; at the same time, we had also added a treatment of imperatives. The fact that the grammar already contained these constructions made the port possible.

There were only some ten constructions that were missing from the grammar. Of these, the most significant was a detailed treatment of the comparative. Fortunately, most of these could be handled (syntactically) by treating the comparative *than* operator as a right adjunct to the word being modified, e.g., *than 12 knots* is a right-modifier of *greater* in *speed greater than 12 knots*. This required only that *than* be treated in the lexicon as a preposition. This certainly does not represent an adequate treatment of the comparative, and indeed, certain complex comparative constructions were not covered by this minimal treatment, for example *Is Puffer's position nearer to OSGP than Queenfish's location is?*.

Other additions to the grammar included:

1. A treatment for *what if* questions, based on the existing treatment of wh-expressions.
2. A treatment for prepositionless time expressions, e.g., *Monday* or *September 4*, etc.
3. A change to allow determiners to have left modifiers, as in *half the fuel* or *only these*.
4. A change to allow adjectives to have a certain class of left modifiers, as in *last three minutes*.
5. A change to allow multiple right noun adjuncts, as in *problems for Fanning that affect mission areas*.
6. A change to allow a preposed nominal argument to an adjective, as in *harpoon capable*.
7. A change to allow fraction expressions (e.g., *two thirds*).
8. Domain specific changes to handle degree expressions and the particular forms of dates encountered in the corpus.

These changes, coupled with a few changes to the restrictions, were sufficient to cover a very substantial portion of the corpus. Constructions that we did not cover (but which would require only modest grammar extensions to cover) include:

1. *or* + *comparative* as a right-modifier of comparative adjectives, e.g., *m5 or lower*.
2. Certain combinations of right noun adjuncts, e.g., *cruisers that are in the Indian Ocean that went to c2 August twenty*.
3. Questions containing the form *how* + *adjective* (*how bad*) and *how* + *adverb* (*how soon*). This hole accounted for a substantial portion of of the incorrectly parsed sentences.

SELECTION

One way to constrain the search space that results from a broad-coverage grammar and lexicon is to apply semantic constraints. Although we did not perform a deep semantic analysis, we did apply shallow semantic (selectional) constraints, to filter out semantically anomalous parses, in a second experiment. This procedure used PUNDIT's Selection Pattern Query and Response (SPQR) component [Lang1988]. We first used SPQR in acquisition mode, to collect semantic patterns, These patterns were then used to constrain search in parsing the test sentences.

The acquisition procedure queries the "domain expert" during parsing, whenever it finds a new pattern, such as a new subject-verb-object pattern, or a new adjective-noun pattern. The expert declares that the pattern is valid, allowing parsing to continue, or that the pattern is invalid, which causes backtracking to find a different analysis (and associated pattern). Information about valid and invalid patterns is stored in a pattern database; as the parser generates each phrase, it checks

the pattern database to see whether the expert has ruled on this pattern; if the user has already classified the pattern, then the user need not be queried again. Thus the system "learns" as it parses more sentences. Following the acquisition (or training) phase, the system can be run in one of two modes: allowing any unknown pattern to succeed (which will overgenerate, assuming that the set of patterns is incomplete), or forcing unknown patterns to fail, which will undergenerate.

To try to obtain maximum coverage of patterns, we generalised the patterns to *semantic class* patterns, rather than patterns of actual words. For example, the subject-verb-object word pattern

[Yorktown,decrease,speed],

can be generalised (using the taxonomy provided by the knowledge base) to the semantic class pattern (the suffix *_C* stands for concept):

[platform_C,change_C,transient_ship_attribute_C].

Previous experience had shown that use of word-level selectional patterns reduced the search by 20%, and the number of parses by a factor of three. We had hoped to achieve greater generality by use of the generalised semantic class patterns. However, due to time constraints, we were only able to process the first 100 training sentences, from which we collected some 450 patterns. This turned out (not surprisingly) to be far too small a set to generate any useful constraints in parsing. We therefore plan to complete our pattern collection on the full training set and rerun our experiment. This should provide us with a good measure of two things: the amount of pruning provided by application of shallow semantic constraints; and the amount of data that is required to obtain a complete set of patterns.

THE KNOWLEDGE BASE

Our experiment with generalization of semantic patterns required the use of a class hierarchy residing in a knowledge base. To support selection, we constructed a first pass at a knowledge base for the resource management domain. The KB contained some 750 concepts. One interesting observation that resulted from this exercise was that the semantic classes required for selection are not necessarily those classes that a knowledge engineer would develop as part of a domain model. In particular, certain words may exhibit similar distribution linguistically (e.g., *average* and *maximum*) but may not necessarily be collected under a single concept to permit easy generalization. For this reason, we may move to a more data-driven paradigm for building the knowledge base in our subsequent experiments.

THE METHODOLOGY

As previously stated, we added domain-independent rules to the grammar, and domain-independent entries to the lexicon, to cover the major constructions observed in the resource management corpus. We then trained on a (subset of) this corpus. The training involved parsing the first 200 sentences and examining and fixing parsing problems in these 200 sentences. We were able to collect semantic patterns only for the first 100 sentences. In both cases, this represents only a small fraction of the available training data (791 sentences). The sentences (training and test) were run on PUNDIT, under Quintus Prolog 2.2 on a Sun 3/60 with 8 MB of memory.

Because PUNDIT normally produces many parses, especially when run without selectional constraints, we allowed the system to run to a maximum of 15 parses per sentence. We report several results below, for purposes of comparison with other groups presenting parsing results. The first result is the number of sentences *obtaining a parse*. We believe that this is not a meaningful figure, however, since it is possible for a sentence to obtain a parse, but never to obtain a *correct* parse. For

this reason, we report a second result: the number of sentences obtaining a *correct parse* within the first 15 parses. In some cases, the system obtained a parse, but did NOT obtain the correct parse within the first 15 parses. In this case, we report it a NOT GETTING A CORRECT PARSE.

Our criteria for counting a parse correct were generally very stringent, and also required obtaining the correct regularised syntactic expression (or ISR). Our criteria included, for example: correct scoping of modifiers under conjunction; correct attachment of prepositional phrase and relative clause modifiers; and correct analysis of complex verb objects.

RESULTS

The table below shows the results obtained with parsing alone (no selectional constraints). We did not report the results obtained from selection, because it turned out that, given our very limited collection of patterns, selection failed to change the test results significantly. However, we plan to collect patterns for the entire training set and rerun this portion of the experiment.

There are several things worth noting in these results. First, the system is quite fast, even running to 15 parses: the average parse time to the correct parse is under 10 seconds for sentences averaging about 10 words/sentence. Second, although the correct parse appears on the average in the third parse, the first parse is correct more than 40% of that time. By adding semantic constraints, we expect to improve that figure substantially, thus driving down further the time to obtain the correct parse.

FUTURE DIRECTIONS

There are several directions that we plan to pursue. The first is to complete our experiments using selectional constraints to prune parses. A second general area that we will focus on over the next few months is the notion of how to *train* the system, that is, using the training set to customize the system to the given domain automatically. In particular, we plan to experiment with a "minimal" lexicon, to determine if we can improve our results by pruning out unneeded syntactic class information (e.g., just having *alert* entered as a noun for this domain). If pruning the lexicon improves our performance significantly, then we will experiment with various ways to use the

	Training (200 sentences)	Test (200 sentences)
Get A PARSE	94%	92%
Get A CORRECT PARSE using SYNTAX only	85%	76%
avg. # of correct parse	2.9	2.6
avg. # of parses/sentence	7.1	6.2
avg. secs. to correct parse	7.5	4.9
avg. secs. total	25.5	17.8

Parsing Results for the Resource Management Domain

training data to tune the system (in some automatic way) to "specialise" the lexicon to the particular application. Similarly, we plan to investigate techniques for using the training corpus to tune the parser to a new domain.

Our ultimate objective is to couple PUNDIT to a speech recognition system. To achieve this, we must focus not only on obtaining the correct parse, but on ruling out incorrect parses. So far, most development work has focused on extending coverage, and not on tightening the grammar to prevent overgeneration. Clearly, it is critical to address this problem if we plan to use a broad-coverage natural language system for spoken language understanding. This will also include developing metrics to measure overgeneration.

Finally, we expect to add the rules to support the in-depth semantic coverage that we have produced for our message domains. Overall, we are optimistic that by adding semantic constraints, plus extending the syntactic coverage in some quite limited ways, we will be able to exceed a 90% correct analysis rate on the test data, which brings the system within the bounds of a realistically useful system.

REFERENCES

[Hirschman1989]
Lynette Hirschman, Martha Palmer, John Dowding, Deborah Dahl, Marcia Linebarger, Rebecca Passonneau, Francois Lang, Catherine Ball, and Carl Weir, The PUNDIT Natural Language Processing System. In *Proc. of the Conference on Artificial Intelligence Systems in Government*, Washington, D.C., March 1989.

[Hirschman1988]
L. Hirschman, A Meta-Treatment of Wh-Constructions in English. In *Proc. of META88, Meta-Programming in Logic Programming*, Bristol, UK, June 1988.

[Lang1988]
F.-M. Lang and L. Hirschman, Improved Parsing Through Interactive Acquisition of Selectional Patterns. In *Proc. of the Second Conference on Applied Computational Linguistics*, Austin,TX, February, 1988.

[Linebarger1988]
M. Linebarger, D. Dahl, L. Hirschman, and R. Passonneau, Sentence Fragments Regular Structures. In *Proc. of the 1988 Annual Conference on Computational Linguistics*, Buffalo, NY, June 1988, pp. 7-16.

Appears in Logic Programming 1, the Proceedings of the Fifth International Conference and Symposium, edited by Robert A, Kowalski and Kenneth A. Bowen

OR-PARALLEL SPEED-UP IN NATURAL LANGUAGE PROCESSING: A CASE STUDY[1]

Lynette Hirschman, William C. Hopkins and Robert C. Smith

Paoli Research Center,
UNISYS Defense Systems
P. O. Box 517, Paoli, Pennsylvania 19301, U.S.A

ABSTRACT

We report on a series of simulation experiments for a large-scale natural language processing system. The results indicate that an or-parallel, all-solutions search provides substantial speed-up (20-30 fold) for this application. Longer sentences also show greater speed-up, giving parse times that increase *linearly* with sentence length. These results were obtained using a simple, application-specific model of independent, non-communicating or-processes in a shared memory environment. Simulations were run with a range of overhead costs; they show that significant benefits are obtained from parallel processing with overheads ranging as high as the median process size, although our estimates of overhead times are substantially smaller than median process size.

1. Introduction

We report here on a series of simulation experiments for a large-scale natural language processing system; our goal was to determine whether processing times in a large-scale application could be reduced substantially by selective exploitation of or-parallelism in a shared-memory environment. We were particularly interested in the granularity of process duration relative to overhead and in the distribution of opportunities for parallelism. We chose a simple, application-specific model of an all-solutions or-parallel search implemented as independent, non-communicating processes in a shared memory environment.

The simulation results show substantial speed-up for this application: at 50% processor effectiveness (speed-up factor

divided by number of processors), the speed-ups are typically in the 20-30 fold range. The longer the sentence, the greater the speed-up, giving parallel parse times that increase *linearly* with sentence length. Simulations were run with a range of overhead costs; they show that significant benefits are obtained from parallel processing with overheads ranging as high as the median process duration, although our estimates of overhead times are substantially smaller than median process duration.

We chose our application (a top-down parser with a broad-coverage English grammar) because it represented a large-scale logic program written in Prolog, developed independently of any notion of parallel processing. We limited our initial experiments to one particular type of application level or-parallelism: parallel exploration of disjunction (alternatives) in grammar rules, although there are many other opportunities for parallelism in the application.

2. Background

There have been a numbe of approaches for the implementation of or-parallelism, including the "Gigalips" project collaboration at Argonne Labs, the University of Manchester and the Swedish Institute of Computer Science [2, 4, 16, 18, 7], work at various institutions in Japan [12, 17], and research at a number of other sites [3, 19]. Or-parallelism has also been addressed within the model of Concurrent Prolog [15]. The goal of these researchers has been to support a transparent port of an application to a parallel Prolog (possibly with minimal annotation to indicate which goals should be run in parallel).

The goals of our research have been focused on finding a useful granularity of parallel in large-scale applications, rather than on the underlying or-parallel implementation. We therefore adopted a simple model of independent, non-communicating or-parallel processes, to minimise issues such as shared variable bindings. This model resembles the PIE model [12], and has some similarities with Shapiro's all-solutions approach in Flat Concurrent Prolog[8]. The limited model of parallelism outlined here would also be appropriate for application of the Kabu-Wake method [17]. However, the grain-size differs from all of these approaches (except the Argonne model), since we focus on an application-level disjunction for spawning or-processes, rather than the finer-grained Prolog disjunction. Our work also differs from many of these [17, 7], in that process creation is determined

solely by the annotations in the program, not by any run-time considerations. Dynamic splitting of processes on demand requires communication and structural overheads that are not required in our simpler model. The compensating cost is the overhead of handling more processes than may be necessary to keep the system busy.

One aspect of our application deserves mention, namely parsing to all solutions. Support for the all-solution approach to parsing becomes particularly important when applying broad-coverage grammars to speech understanding, where there is substantial indeterminacy in the input signal, necessitating massive exploration of parallel paths during parsing. The importance of the speech processing application alone would justify special architectures tailored for parallel parsing, although clearly use of an application-independent model of parallelism would be both more general and more flexible.

Our all-solutions application explains why our results show substantial parallelism, even with an overhead cost comparable in size to average process duration; it may also explain why our results differ from the negative results reported by Fagin and Despain [6] who looked only at single solution search in most cases, and found, not surprisingly, little speed-up due to or-parallelism.

3. Description of the Application

Our experiments are based on the parser and grammar of the Unisys PUNDIT[9] natural language processing system (Restriction Grammar) [9, 13]. The grammar consists of context-free BNF definitions, augmented with constraints, in the style of a logic-grammar. However, unlike Definite Clause Grammar [14] or its variants, there are no explicit parameters in Restriction Grammar. Contextual information is stated as well-formedness constraints on the parse tree, which can be freely traversed by a special set of "restriction operators" which examine the shape of the parse tree. To support the free tree traversal, the parser maintains a complex data structure consisting of both the parse tree and a path up to the root of the tree (for traversal up the tree), making grammar rule execution more complex (and larger grained) than a comparable operation in a plain DCG. The Restriction Grammar parser is implemented by an interpreter (instrumented for debugging) and a dynamic translator (for parsing with a stable grammar) [5].

We used the Restriction Grammar interpreter as the basis for the simulation experiments, because it supported the instrumentation required to generate input to the simulator. In order to obtain timing data to support the simulations, we modified the grammar interpreter slightly to use a continuation-based interpreter. At each grammar rule disjunction, the interpreter constructs an explicit continuation to be passed to the new process. Thus each disjunction becomes a self-contained process which runs to completion (or failure) completely independently of other processes. All that is required for a process spawn is a "clean copy" of the continuation data structure, with logical variables in this structure named apart from logical variables in other processes. This removed any dependency on the Prolog stack and enabled us to run crude timing experiments on the copying operation.

The English grammar consists of approximately 125 BNF definitions and 55 restrictions. This base grammar is augmented by a meta-rule which generates additional rules for handling conjunction, increasing the size of the grammar by about 50%. The coverage of the grammar is very broad; it includes an extensive treatment of co-ordinate conjunction [10], detailed coverage of

Sample Grammar		
sentence	::=	subject, predicate, {subj_verb_agree}.
subject	::=	noun_phrase
noun_phrase	::=	left_mods, *noun, right_mods.
left_mods	::=	article, adjectives.
right_mods	::=	prepositional_phrase; null.
article	::=	*determiner; null.
adjectives	::=	*adjective, adjectives.
adjectives	::=	null.
predicate	::=	*verb, object.
	::=	*verb, {be_verb}, be_predicate.
object	::=	({transitive_verb}, noun_phrase); ({intransitive_verb}, null).
be_predicate	::=	*adjective; prepositional_phrase.
prepositional_phrase	::=	*preposition, noun_phrase.
null	::=	% Marks empty element.

Table 1.

complex subject and complement types, fine-grained analysis of noun phrases, treatment of relative clauses and questions, and also a small set of definitions for handling fragmentary sentences (e.g., ***Metal particles in oil*** or ***Suspect coupling to be sheared***).

The result of the broad-coverage of the grammar is that parsing involves a good deal of search. There are approximately 350 disjunctions in the grammar (a branching factor of about 3). Many sentences receive multiple parses; even sentences receiving only a single parse still explore many alternative paths that end in failure.

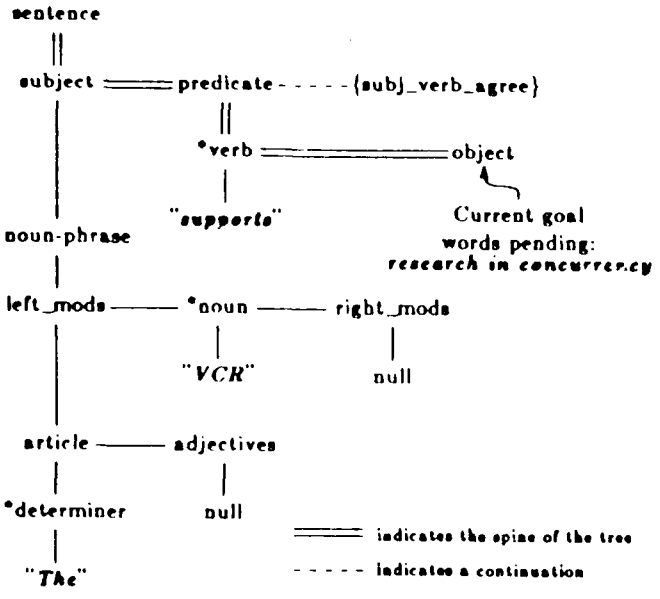
The Restriction Grammar interpreter consists of a clause defining the parser actions for each type of connective that can be found in a grammar rule: conjunction, disjunction, terminal *nodes (indicated by an asterisk), literals, and restrictions (indicated* by curly braces). Table 1 shows a highly simplified grammar, illustrating rules with conjunction (**sentence**, **noun_phrase**, **left_mods**, etc.), and disjunction (e.g. **right_mods**, **article**, **object**). The grammar disjuncts are the level at which we search for all *solutions*.[4] Figure 1 shows the corresponding parse tree for a short sentence.

4. Simulation Models

The experimental approach uses a model-based simulation system named the ***Virtual Computation Recorder*** or ***VCR*** [11], which *infers* the concurrent behavior of the parser from its sequential behavior. Two processing models for the PUNDIT parser drive the simulation effort. The ***sequential execution model*** is derived from sequential Prolog execution, and defines the granularity of the simulations. The ***concurrent execution model*** is a special case of a shared-memory OR-parallel execution model for Prolog. These models and their relationship are a key element of the approach.

The PUNDIT parser searches for all parses of the input string, using standard Prolog backtracking to explore the search tree completely. A single parse tree is built and unbuilt in backtracking, so there is only one partial parse tree, representing the state of the currently active parse, at any time.

The concurrent execution model of the parser supports the concurrent construction of independent parse trees, spawning processes in shared memory for the disjunctive goals encountered. Each child process will attempt to complete the partial parse supplied by its parent using one of the goals. Spawning is under



Partial Parse of
The VCR supports research in concurrency

Figure 1: Example of Parse Tree

the control of a policy which specifies which disjuncts are spawned and which are left to be pursued with the normal failure/backtrack mechanisms. At one policy extreme (never spawning), the model is that of sequential Prolog; at the other (always spawning), no backtracking occurs, as each disjunct is explored by its own process. Failure to satisfy a spawned goal results in termination of the process. The cost of backtracking and of maintaining backtracking information can thus be avoided for spawned goals. For simplicity, the rest of the discussion will discuss only spawned disjuncts. Any unspawned disjuncts are treated with the normal Prolog mechanisms.

The models are not concerned with the details of satisfying a goal beyond identifying subgoals and spawning processes for them. Thus, only four types of processing are identified in the sequential execution of the PUNDIT parser: ***start-goal***, which sets a goal, ***complete-goal***, the successful derivation of the goal, ***fail-goal***, the failure to derive the goal, and ***backtrack***, the search back to the most recent alternative goal. Annotations in the PUNDIT source identify the transitions from one type of processing to the next; these annotations serve two purposes: to generate the behavior description in terms of chunks of processing, which are called ***atoms***, and, in a separate step, to allow timing of the atoms.

Each disjunct represents a distinct attempted parse, and has its own parse tree, which is initially a copy of the parse tree of its parent. The result of a concurrent parse is the forest of parse trees produced by the successful parses.

4.1. Process State A child process receives the parse tree computed so far, the continuation stack (specifying the remaining conjoined subgoals for the goal being sought at each level of the tree), the return path to the root of the parse tree, the current goal, and the remaining input string. No other data from the parent process is needed. The concurrent execution model exploits sharing between parent and child of the parts of this data which contain no unbound logical variables when the child is spawned; only those portions are copied that may contain unbound logic variables, as outlined in Table 2.

As the parse proceeds, the tree is expanded along its right edge. All unbound logic variables are on the spine (the path from the root to the current site of tree-building), so the completed subtrees to the left of the spine are constant, and may be shared. The return path is the inverse of the spine, containing back pointers along the path to the root. The parse tree (Figure 1) for the simplified grammar example identifies the spine and return path with doubled links. The spine itself must be copied, renaming apart the logical variables in it for the child process; a

State component	Shared	Local
parse tree	subtrees left of spine	spine
return path	-	return path
continuation stack	continuation stack	head pointer
remaining words	wordstream	"next" pointer

Table 2: Process State

new return path corresponds to the new spine. The continuation stack contains no logical variables and may be shared.

To determine the cost of performing the necessary copying and naming apart of variables, the spine of the tree and the return path were copied explicitly in Prolog. The spine grows and shrinks during the course of parsing and exact copying time depends on the actual length of the spine when copied. We measured the copying time during an actual parse; the average copying time was found to be about 1 millisecond. This corresponds reasonably well with intuition (approximately 50-100 nodes copied, with one logical inference required per node copied on a 50K Lips Prolog). The copying time increased only slightly with the length of the spine, indicating that other costs were being included; we have, however, taken the conservative path of using the measured copying times. The time to start up the spawned process is arbitrarily set at 1 millisecond to allow for process creation and system overheads. All these timings assume a simple model of parallel Prolog in which the specialised copying is not supported as a primitive; directly implementing such a primitive could be expected to reduce the copying time substantially.

Test Sentences	
1.1.1	starting air regulating valve failed.
4.1.1	while diesel was operating with sac disengaged, the sac lo alarm sounded.
4.1.3	pump will not turn when engine jacks over.
5.1.2	disengaged immediately after alarm.
8.1.2	inspection of lo filter revealed metal particles.
9.1.1	sac received high usage during two becce periods.
9.1.4	loud noises were coming from the drive end during coast down.
22.1.1	loss of lube oil pressure during operation.
25.1.3	suspect faulty high speed rotating assembly.
28.1.1	unit has excessive wear on inlet impellor assembly and shows high usage of oil.
31.1.3	erosion of impellor blade tip is evident.
31.1.4	compressor wheel inducer leading edge broken.

Table 3.

5. Analysis of Results

We performed a large number of experiments, varying the sentence being parsed, the number of processors available, the times for processing atoms, and most important, the overhead associated with spawning a new process. The sentences are listed in Table 3; this discussion will use the results for a typical sentence, *Unit has excessive wear on inlet impeller assembly and shows high usage of oil.* Full results, with extensive data, are available in [8]. All results here assume a process spawn on every grammar rule disjunction.

5.1. Concurrency and Granularity Figure 2 shows the concurrency discovered in the parsing of a typical sentence; it graphs the number of active processes over time, assuming enough processors (116) so that one is available to run each process when it is spawned. As is evident in the example, the concurrency is irregular, with one or more peaks and potentially long start-up and shut-down tails. Furthermore, the shape of the

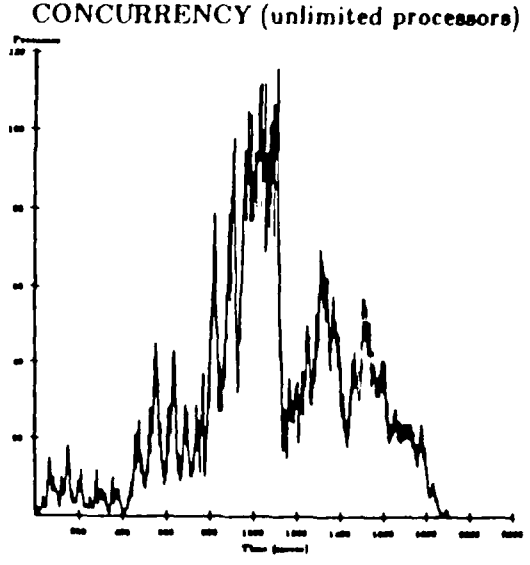


Figure 2: Typical Unconstrained Concurrency

concurrency varies markedly from sentence to sentence, depending on where in the sentence multiple parses are attempted and how long the parser follows a path before the path ends in failure.

Figure 3 shows the active process distribution for the same sentence with a 50 processor limit, which increases the parse time by about 10% from the unlimited case. Processes that are spawned when all processors are busy wait in a strict (FIFO) queue. The effect of varying the number of processors on speed-up is shown in Figure 4; the curves have a characteristic shape with a distinct knee that reflects the decreasing effectiveness of additional processors. Below the knee, speed-up is close to linear; beyond it, the waiting queue empties and additional processors are increasingly idle.

The duration of the processes and the relative cost of spawning them are extremely significant for matching the processing to a multiprocessor architecture. Our results here are quite encouraging; the median process time, using measured atom

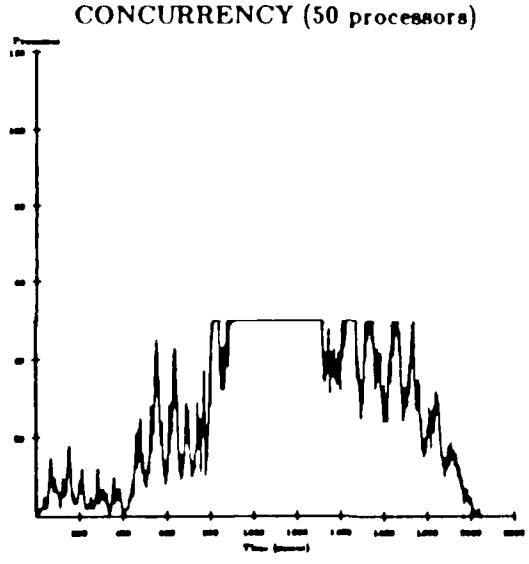


Figure 3: Typical Constrained Concurrency

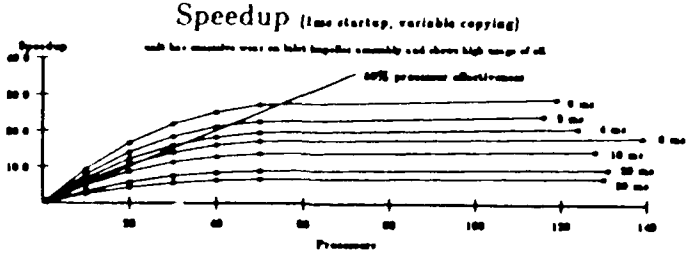


Figure 4: Speed-up Curves

timings from interpreted Prolog runs, is typically around 9 milliseconds. Figure 5 shows the distribution of process duration in 5 millisecond increments; it is remarkably invariant over the sentences tested.

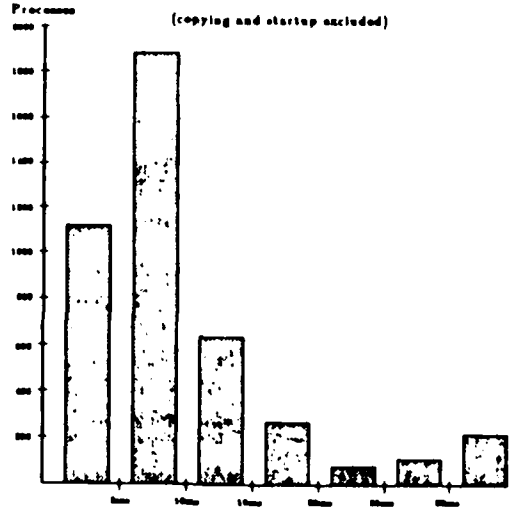


Figure 5: Process Length Distribution

5.3. Speed-up The calculation of parallel speed-up factors, especially the selection of the base time, is notoriously open to "optimization of results." The generally accepted rule is to compare to the best sequential implementation, to get a fair measure of system design choices. We take the sequential parse times of the unmodified PUNDIT parser as our standard. (The instrumentation technique required using a grammar interpreter written in PROLOG; an additional doubling of speed is obtainable by translating the grammar directly to PROLOG. Section 6 addresses the implications of this speed-up.) The timings are *reconstructed from the measured behavior* and atom timings of the parser (as modified for continuation stacking) and to normalize for the systematic effect of the measurements. The effect of the parser modifications and the instrumentation is to increase the execution time by about 50%; to compensate, we increase the copying overhead (by more than 50%) to 2 milliseconds to maintain the proper (conservative) ratio. All timings are derived from Quintus Prolog 1.5 running on a Sun 3/160 with 16 megabytes of memory.

Figure 4 shows a family of speed-up curves for the example sentence. The speed-up factor over the sequential execution time is shown as a function of the number of processors available. The additional parameter, which differs among the several curves, is the *copying overhead*, discussed in the next paragraphs. The straight line identifies 50% processor effectiveness, the points at which the speed-up factor is half the number of processors[5]. This arbitrary limit is taken as the threshold for acceptable system performance; it typically is at or near the knee of the speed-up curve. Table 4 summarizes the speed-up for the sentences parsed.

5.3. Effect of Overhead Figure 4 shows the speed-up curves for our example sentence under five different copying overheads, ranging from 0 to 30 milliseconds; the process start-up is constant at 1 millisecond. As can be seen, if the copying time exceeds 4 milliseconds, more than a third of the potential speed-up is lost; at *10 milliseconds, more* than half is gone. This suggests that parallelism can be effectively exploited if process overhead is kept to a fraction of the median process duration (here, 9 milliseconds).

Shen and Warren report simulation results [16] that show the same characteristic curves (their Figure 3) using an artificial time unit of a "resolution time", with the cost of process

PUNDIT Concurrency Results (time copying, time startup)

Sent.	lth.	time(sec.) seq.	time(sec.) par.	maximum spd.	maximum proc.	50% effect. spd.	50% effect. proc.
5.1.2	4	10.8	0.9	11.8	82	-	-
1.1.1	5	7.3	0.7	10.1	42	9.9	20
25.1.3	6	26.8	1.5	18.5	121	-	-
31.1.4	6	24.9	1.1	23.2	71	-	-
31.1.3	7	6.1	0.9	7.1	28	6.9	14
6.1.2	7	7.8	0.9	8.5	37	7.9	16
22.1.1	7	22.5	1.9	11.8	59	-	-
4.1.3	8	7.4	1.2	6.4	37	5.9	12
9.1.1	8	13.7	1.2	11.9	42	11.6	23
9.1.4	11	37.3	1.4	26.1	93	22.6	45
4.1.1	12	51.5	1.8	31.8	163	27.8	56
28.1.1	14	46.2	1.9	24.3	116	21.1	42

Table 4: Summary of Speed-up Results

spawning usually being four resolutions (some tests vary it from 0 to 16 resolutions). They also find that the overhead has a substantial effect on the achievable speed-up. This leads us to hypothesize that the key parameter relating overhead to speed-up is the ratio of process initialization overhead to median process duration; this remains to be confirmed by additional experimentation.

5.4. Form of the Concurrency Spawning separate processes for grammatical disjuncts usually has the effect of starting the exploration of the "correct" parse sooner than would happen sequentially (unless it is the left-most; the grammar has indeed been written to encourage this behavior, which is desirable in a sequential system). Spawning all disjuncts has the effect that for any potential parse, there is a series of processes that attempts that parse with no unnecessary searching. Given sufficient resources, this will accomplish the ultimate goal of reducing the time to develop the correct parse.

5.5. Effect of Sentence Length Figure 6 shows the relationship between sentence length and parallel parsing time with unlimited processors. For these thirteen sentences, the relationship appears to be linear, which is consistent with the processing model. Since each potential parse is performed as quickly as possible, the time to complete it is proportional to the number of nodes in the parse tree; this is very close to the number of words

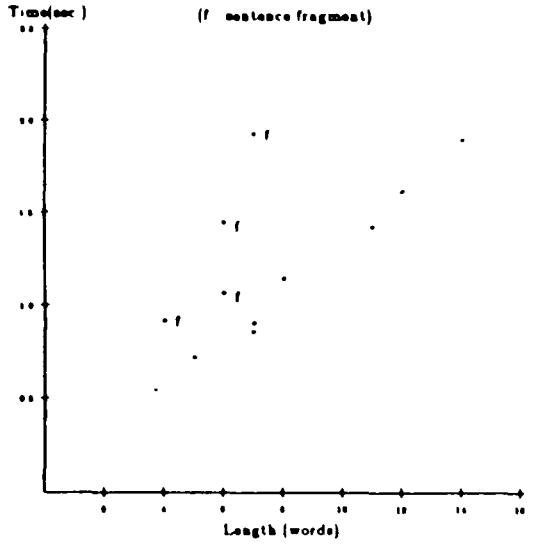


Figure 6: Linear Parse Time

(being at worst *N log N* for *N* words).

5.6. Linguistically Guided Concurrency Our original intent was to investigate the use of static linguistic knowledge to guide spawning, e.g., spawn only on certain "rich" nodes or nodes with a high probability of success. However, the results from the simpler always-spawn policy were sufficiently good that we have deferred investigation of other policies. As process duration is reduced, however, we may return to this line of investigation as a way of increasing process granularity relative to overhead.

6. Conclusions

We have demonstrated exploitable or-parallelism in an important real Prolog application, and have simulated significant speed-up with an application-specific parallel execution model. For a small but representative set of sentences, we have observed linear-time parsing in the unlimited-processor case.

Some necessary work remains to sharpen and improve the simulation model and parameters. We have previously

mentioned the instrumentation overhead that remains in the atom duration measurements. We have also not treated garbage collection properly, as we have not yet dug into the Prolog implementation to identify and measure it precisely to allow prediction of a multiprocessor equivalent. The overhead of maintaining choice points and trail information, unnecessary for spawned disjuncts, is another unknown. These effects may reduce the granularity of the processes, making the overhead more significant if it is not further optimised. Finally, the measurements are made on a parser that interprets the grammar, rather than running the more efficient version of the grammar obtained by translating it to PROLOG. This typically doubles the parser speed, so speed-up projections for this case will be consistent with the results reported here for the 4 millisecond copying cost.

The results described here apply to an application-specific large-grained form of concurrency. Research in related areas [1, 16] points to the existence of finer-grain concurrency within the general or-parallel model that would apply to the computations that we have taken as atomic. Further research is necessary to determine the amount of additional speed-up that can be gained by combining approaches.

We have recently begun collaboration with the researchers at the Swedish Institute of Computer Science; our plan is to furnish the application described here, for benchmarking on the Aurora Or-parallel machine. This work should provide us with the ability to validate our simulation results. It should also provide insight into speed-ups obtainable on large-scale applications in a general or-parallel implementation.

NOTES

1. This work has been supported in part by DARPA under contract F30602-86-C-0093, administered by Rome Air Development Center, in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research, and in part by National Science Foundation contract DCR-85-02205. APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

2. In an all-solutions environment, or-parallelism merges with and parallelism, because exploration of alternatives becomes mandatory: alternatives become indistinguishable from independent, non-communicating and-processes.

3. ***Prolog Understands Integrated Text***

4. There are many additional opportunities for or-parallelism, as well as for independent and-parallelism that we have not exploited in this

simulation; this will be the subject of future work.

5. Processor utilisation, which merely measures how busy the processors are, makes an apparent virtue of overhead. Processor effectiveness, sometimes called *efficiency*, is the average *useful* processor utilisation.

REFERENCES

[1] T. Blenko, Compiling Logic Programs to Applicative Form, LBS Technical Memo 74, Paoli Research Center, Unisys Corporation, Paoli, PA, January, 1988.

[2] A. Ciepielewski, S. Haridi, and B. Hausman, Initial Evaluation of a Virtual Machine for OR-Parallel Execution of Logic Programs. *Proceedings of IFIP TC-10*, U. of Manchester , 1985.

[3] J. S Conery, Binding Environments for Parallel Logic Programs in Non-Shared Memory Multiprocessors. In *Proc. of the 1987 Symposium on Logic Programming*, San Francisco, 1987, pp. 457-467.

[4] Terry Disz, Ewing Lusk, and Ross Overbeek, Experiments with Or-Parallel Logic Programs. In *Proc. of the Third International Conference on Logic Programming*, J. L. Lassez (ed.), Melbourne, 1987, pp. 576-600.

[5] John Dowding and Lynette Hirschman, Dynamic translation for Rule Pruning in Restriction Grammar. In *Proc. of the Second International Conference on Natural Language Understanding and Logic Programming*, Vancouver, August, 1987.

[6] B. S. Fagin and A. M. Despain, Performance Studies of a Parallel PROLOG Architecture. In *Proceedings of the 14th International Symposium on Computer Architecture*, Pittsburgh, Pennsylvania, June, 1987, pp 108-116.

[7] B. Hausman, A. Ciepielewski, and S. Haridi, OR-Parallel Prolog Made Efficient on Shared Memory Multiprocessors. In *Proc. of the 1987 Symposium on Logic Programming*, San Francisco, 1987, pp 69-79.

[8] Lynette Hirschman, William C Hopkins, and Robert C. Smith, Simulation of Or-Concurrency in the PUNDIT Parser, LBS Technical Memo, Paoli Research Center, System Development Corp., Paoli, PA, June, 1988.

[9] L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985, pp. 244-261.

[10] L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* 4, 1986, pp. 299-328.

[11] W. Hopkins and R. Smith, Concurrency Simulation by Abstract Interpretation, LBS Technical Memo, Paoli Research Center, Unisys Corp., Paoli, PA, December, 1987.

[12] T. Moto-oka, H. Tanaka, H. Aida, and T. Maruyama, The Architecture of a Parallel Inference Engine - PIE. *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1984, pp. 479-488.

[13] Martha S. Palmer, Deborah A. Dahl, Rebecca J. Schiffman , Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information. In *Proc. of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, August 1986.

[14] F. C. N. Pereira and D. H. D. Warren, Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13, 1980, pp. 231-278.

[15] Ehud Shapiro, An Or-Parallel Execution Algorithm for Prolog and its FCP Implementation. In *Proc. of the Third International Conference on Logic Programming*, J.-L. Lassez (ed.), Melbourne, Australia, 1987, pp. 311-337.

[16] K. Shen and D. H. D. Warren, A Simulation Study of the Argonne Model for Or-Parallel Execution of Prolog. In *Proc. of the Third International Conference on Logic Programming*, J.-L. Lassez (ed.), Melbourne, Australia, 1987, pp. 56-68.

[17] Y. Sohma, K. Satoh, K. Kumon, H. Masusawa, and A. Itashiki, A New Parallel Inference Mechanism Based on Sequential Processing, TM-0131, ICOT, 1985.

[18] D. H. D. Warren, The SRI Model for Or-Parallel Execution of Prolog -- Abstract Design and Implementation. In *Proc. of the 1987 Symposium on Logic Programming*, The Computer Society of the IEEE, San Francisco, CA, 1987, pp. 92-102.

[19] H. Westphal and P. Robert, The PEPSys Model: Combining Backtracking, AND- and OR-Parallelism. In *Proc. of the 1987 Symposium on Logic Programming*, San Francisco, 1987, pp. 436-448.

THE PUNDIT NATURAL-LANGUAGE PROCESSING SYSTEM

Lynette Hirschman
Martha Palmer
John Dowding
Deborah Dahl
Marcia Linebarger
Rebecca Passonneau
Francois-Michel Lang
Catherine Ball
Carl Weir

Reprinted from PROCEEDINGS OF THE ANNUAL AI SYSTEMS IN GOVERNMENT CONFERENCE, Washington, D.C., March 27-31, 1989

The Computer Society of the IEEE
1730 Massachusetts Avenue NW
Washington, DC 20036-1903

Washington • Los Alamitos • Brussels

COMPUTER SOCIETY PRESS

The PUNDIT Natural-Language Processing System

Lynette Hirschman, Martha Palmer, John Dowding,
Deborah Dahl, Marcia Linebarger, Rebecca Passonneau,
François-Michel Lang, Catherine Ball, Carl Weir

Paoli Research Center, Unisys
P.O. Box 517, Paoli, PA 19301

ABSTRACT

In this paper we describe the PUNDIT[1] text-understanding system, which is designed to analyze and construct representations of paragraph-length texts.[2] PUNDIT is implemented in Quintus Prolog, and consists of distinct lexical, syntactic, semantic, and pragmatic components. Each component draws on one or more sets of data including a lexicon, a broad-coverage grammar of English, semantic verb decompositions, rules mapping between syntactic and semantic constituents, and a domain model. Modularity, careful separation of declarative and procedural information, and separation of domain-specific and domain-independent information all contribute to a system which is flexible, extensible and portable: Versions of PUNDIT are now running in five domains, (four military and one medical).

1 Introduction

PUNDIT is a large, modular natural-language text-understanding system implemented in Prolog. This paper discusses the design and implementation of PUNDIT, focusing on several issues which have been crucial to the development of the system:

- Portability, supported by clear factoring of domain-independent and domain-specific information, and by a collection of tools to support creation of the various modules;
- Modularity, supporting incremental development of a large-scale system including lexical, syntactic, semantic, and pragmatic components;
- Integration of multiple sources of information, to provide search focus during parsing and convergence on a correct interpretation of individual sentences (and ultimately of the entire discourse);
- Robustness, provided by a broad-coverage grammar, integration of multiple knowledge sources to detect inconsistent information, and feedback to the user to provide help in diagnosing missing or incorrect information;
- A development environment for the construction of a large-scale natural-language processing system, including tools for debugging, testing, updating, and tailoring the system to different types of development.

We begin by describing some applications of PUNDIT and the domains of discourse in which PUNDIT has been used. Next, we present the three principal modules of the PUNDIT system: Syntax, semantics, and pragmatics. Section 6 deals with the problem of portability, and we conclude with a brief discussion of implementation and system issues. Figure 1 describes the overall architecture of the system.

2 Applications and Domains

The core facilities of PUNDIT, which are represented by rectangular boxes in Figure 1, are domain- and application-independent: They provide the basic text-understanding capabilities of the system, accepting natural-language text

[1] Prolog UNDerstanding of Integrated Text.

[2] This work has been supported in part by DARPA contract N00014-85-C-0012, administered by the Office of Naval Research, and in part by National Science Foundation contracts DCR-8202397 and DCR-85-02205, as well as by Independent R&D funding from System Development Corporation, now part of Unisys Corporation. APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

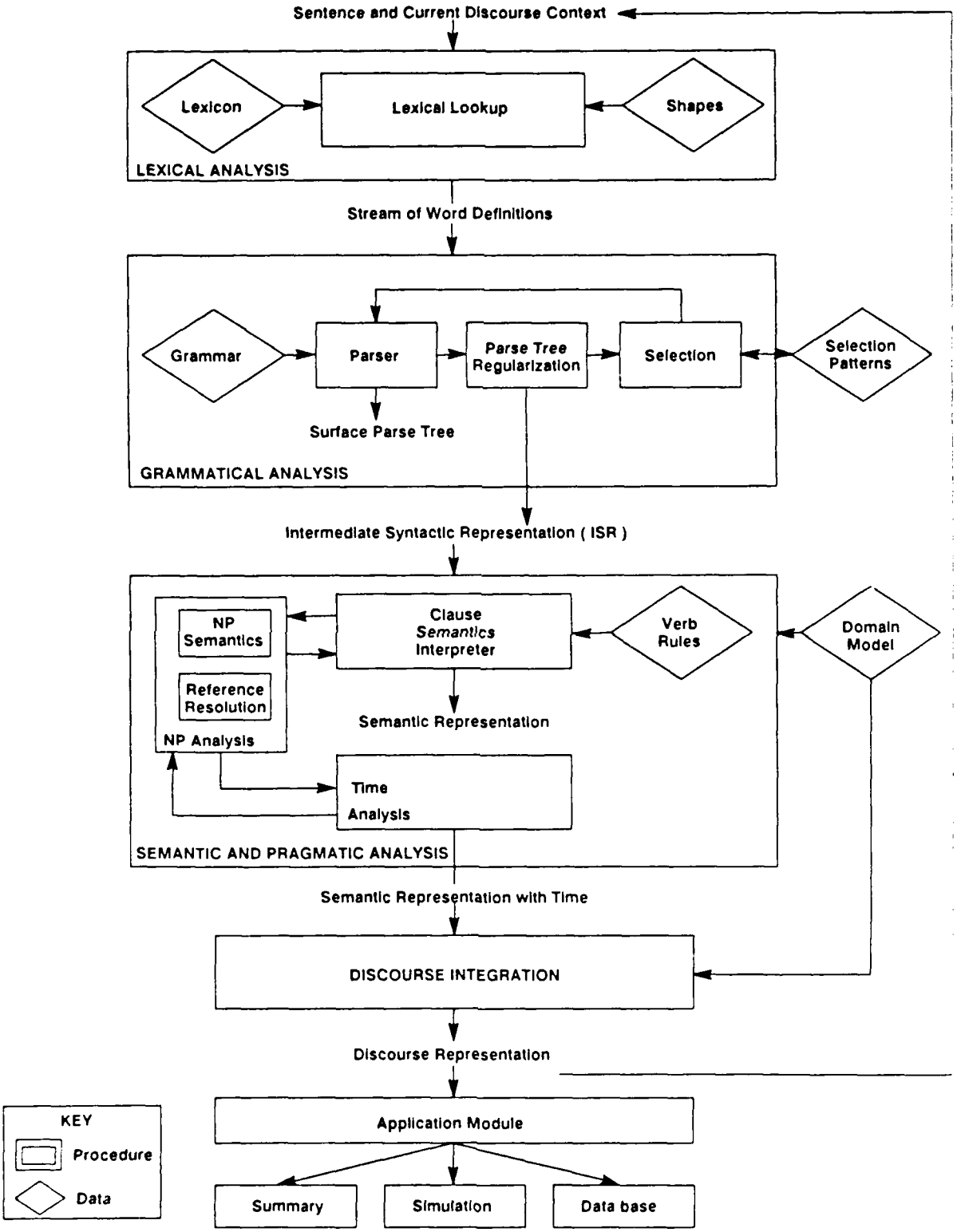


Figure 1: Organization of PUNDIT

as input and producing a representation of the meaning of the text as output. (See Figures 2 and 3 for sample output.) For a specific application within a given domain, the core system is augmented by domain-specific data files (the knowledge base, the lexicon, etc.) and by application modules for special front-end and back-end functions. These provide the user interface to PUNDIT and may perform application-specific tasks based on PUNDIT's output. Some of these tasks include summarizing the input, interfacing to an expert system, and updating or querying a database. The two major application areas to date have been message processing and database queries. Future plans include integration of a speech-recognition capability.

2.1 Message Processing

Applications have been developed to process four types of messages or reports: maintenance reports on Burroughs equipment, messages reporting equipment failures on Navy ships (CASREPs), Navy RAINFORM tactical messages, and *Trouble and failure reports* (TFRs) from Trident submarines. We are now also porting the system to a domain of medical abstracts.

The CASREPs application was developed in the context of a Navy battle management domain focused on force readiness. For this application, the system processes the remarks field of messages describing failures of *starting air compressors* (sacs), and generates a tabular summary of the major problems and findings. A sample CASREP summary is shown in Figure 2.

In the RAINFORM message application we also process message "header" information (originator, report date, etc.), and use this to create a context for the interpretation of the message.

The Trident TFR application, which is currently under development, allows a message to be collected interactively through a prompt-response dialogue. The system asks the user a number of questions designed to elicit key facts about the problem (what went wrong, what was the cause, what action was taken, etc.); the user's answers are analyzed and validated by PUNDIT, and the results of analysis are used to update a historical database of equipment problems. This database can then be queried using a database query language or an English-language front end (another application of PUNDIT—see subsection 2.2). The three Navy applications have been developed as part of the DARPA Strategic Computing program.

2.2 Natural Language Query Processing

The core components of PUNDIT analyze an English query and produce a set of meaning representations, which are then mapped to a set of database relations. For queries to a foreign database (either remote or local), the database relations are translated into QUEL, and the resulting QUEL query is used to access an INGRES database. The results are displayed to the user. Alternatively, the database relations may be used to access an internal Prolog database. These approaches have been used to support English queries on an INGRES database of ship movements, and on Prolog databases of CASREPs, RAINFORMs, and computer failure reports.

3 The Syntax Component

In PUNDIT, syntactic analysis yields two syntactic descriptions of the sentence. One is a detailed surface parse tree, and the other is a regularized structure called the *Intermediate Syntactic Representation* (ISR). The surface structure parse tree is produced during syntactic analysis, and is used to check syntactic constraints. The ISR is constructed incrementally from the surface parse tree and reduces surface structure to a canonical predicate-argument form appropriate for input to semantics (see Section 4) and selection (see Section 6.1).

3.1 The Grammar

The syntactic component of PUNDIT is based on the logic-grammar formalism of Restriction Grammar [8] (a descendent of Sager's string grammar [16]), and includes facilities for writing and maintaining large natural-language grammars. The grammar consists of a set of context-free BNF definitions (currently numbering ap-

message:
LOSS OF LUBE OIL PRESSURE DURING OPERATION.
INVESTIGATION REVEALED ADEQUATE LUBE OIL
SATURATED WITH METALLIC PARTICLES.
REPLACED SAC .

```
Finding:
Part: oil pressure        State: lowered

Finding:
Part: oil                 State: saturated with particles

Status of Sac:
Part: old sac             State: removed from starting_air_system

Status of Sac:
Part: new sac             State: included in starting_air_system
```

Figure 2: CASREP summary

proximately 80) augmented by restrictions (approximately 35), which enforce context-sensitive well-formedness constraints and, in some cases, apply optimization strategies to prevent unnecessary structure-building. The grammar can either be interpreted or translated (or a mixture of both), and uses a top-down left-to-right parsing strategy augmented by dynamic rule pruning for efficient parsing [4]. In addition, a meta-grammatical component generates definitions for a full range of co-ordinate conjunction structures [6] and *wh*-constructions [7]. Syntactic phenomena treated by the current grammar include declarative sentences, imperatives, questions, sentence fragments[3] [10], sentence adjuncts, conjunction, relative clauses, complex complement structures, *wh*-constructions, and a wide variety of nominal structures, including compound nouns, nominalized verbs and embedded clauses.

3.2 Intermediate Syntactic Representation

In parsing a sentence, the syntax processor uses the rules of the grammar to produce two structures: a detailed surface-structure parse, and an *Intermediate Syntactic Representation* (ISR), which is a regularization of the syntactic parse into a canonical predicate-argument form. Regularization is accomplished by annotating each rule in the grammar with an expression in the lambda-calculus-based *translation rule language*, which provides directions for combining the ISRs from the children of a node into the ISR of their parent.

An important step in the regularization process involves mapping fragment structures onto canonical verb-subject-object patterns, and flagging missing elements. For example, a *two* fragment consists of a *tensed verb + object*, such as *Replaced sac*. Regularization of this fragment maps the *two* syntactic structure into the *verb + subject + object* structure *replaced + pro(elided) + sac*. The semantic and pragmatic components provide a semantic filler for the missing subject (represented as `pro(elided)` in the ISR) using general pragmatic principles and specific domain knowledge [12]. The ISR for the sentence *Re-*

[3]The rules for fragments enable the grammar to parse the "telegraphic" style characteristic of message traffic, such as *Disk drive down*, and *Has select lock*.

placed sac is the following:

```
[past, replace,
  subj([pro([elided,singular,ID1])]),
  obj([det([]),
       [noun([sac,singular,ID2])]])]
```

The variables ID1 and ID2 in the ISR are instantiated by the semantics and reference resolution components to distinct referential indices.

4 The Semantics Component

4.1 Clausal Decompositions

Semantic representations of sentences are derived from the corresponding ISRs. The main verb of each clause is decomposed into a predicate/argument structure that makes explicit the relations comprising the verb's meaning. The arguments of each predicate in the verb decomposition are either further semantic predicates or *thematic roles* [11]. For example, the verb *replace* decomposes into the following representation:

```
causeP(agent(AGENT),
       becomeP(exchangedP(
                    object1(OBJ1),
                    object2(OBJ2))))
```

The causeP predicate denotes a relation between an individual (the agent) and another predicate representing the type of relation that the agent is responsible for bringing about in a **replace** event. AGENT, OBJ1, and OBJ2 are Prolog variables that are eventually bound to discourse referents through a process of semantic interpretation briefly described below. The first argument of causeP, the **agent** thematic role, thus evaluates to a discourse referent representing the individual doing the **replacing**. The second argument of causeP, namely

```
becomeP(exchangedP(object1(OBJ1),
                   object2(OBJ2)))
```

is a predicate representing the type of situation that is brought about by the **agent** discourse entity. It consists of two nested predicates: the aspectual operator **becomeP**, which provides information about the temporal structure of a **replace** event (cf. Section 5.3 and Figure 3), and the predicate **exchangedP**, which specifies the result of a **replace** event as a relation among the discourse entities that are eventually bound to OBJ1 and OBJ2.

4.2 Thematic Roles

In the domains that PUNDIT currently supports, the thematic roles used are: *actor*, *agent*, *experiencer*, *goal*, *instigator*, *instrument*, *location*, *object1*, *object2*, *patient*, *reference_pt*, *source* and *theme*. A dramatically different domain such as children's birthday parties might require additional thematic roles. Our goal is to establish a collection appropriate for a wide range of domains, since the algorithm that assigns referents to roles is general across domains.

For any specified context, a thematic role may be asserted to be *obligatory* or *essential* [12]. If a role is obligatory in a specified context, then it is required that it be filled by a referent introduced overtly by some expression in the current utterance. If a role is essential, it must be filled either by a referent introduced by an expression in the current utterance, or by some referent in the discourse context. Roles that are not declared to be obligatory or essential with respect to some predicate environment need not be filled at all in that environment.

The algorithm used to fill thematic roles makes use of syntactic mapping rules to associate types of syntactic constituents with thematic roles. For example, one mapping rule indicates that a subject can provide a filler for the *agent* role. These mapping rules can be tailored to account for idiosyncracies of particular verbs, but for the most part they are fairly general and port readily to new domains. Selectional rules express restrictions on fillers of thematic roles for a given class of verbs. These restrictions are more domain specific, since they are heavily dependent on the co-occurrence patterns and semantic classes characteristic of the verb in the given domain.

4.3 Nominalizations

The interpretation of nominalizations is similar to the interpretation of clauses, but differs in

what linguistic information is accessed and how thematic roles are assigned [3]. In the most straightforward cases, the decomposition of a nominalization is the same as that of its corresponding verb, but with different syntactic mapping rules. For example, comparing *Replaced sac* with *Replacement of sac*, a likely filler for the *object1* role of *replace* is the referent introduced by the syntactic object, but a likely filler for the *object1* of *replacement* is the referent introduced by an *of*-prepositional phrase. In addition to having different mapping rules, nominalizations do not have any obligatory roles. A role that would be obligatory for the verb, such as the *object1* role for *replace*, is considered to be essential for nominalizations.

5 The Pragmatics Components

5.1 Reference Resolution

Reference resolution is called by semantics when it is ready to instantiate a thematic role with a specific referent. Reference resolution finds the referent of noun phrases, creates unique identifiers for entities (see the `id` relations in Figure 3), and also establishes other semantic relationships among entities mentioned in the text. The reference resolution component handles the following phenomena:

- pronouns (including zeroes, such as the unexpressed subject in *Replaced sac*) and *one*-anaphora, using a syntax-based focusing algorithm [2];
- definite and indefinite noun phrases, as well as noun phrases without determiners found in telegraphic-style messages;
- *implicit associates*, [2,1] such *sac* and *pressure* in *Sac failure due to loss of oil pressure*, where it is important to express the fact that the oil under consideration is the oil in the sac, not just any oil;
- conjoined noun phrases;
- nominal references to events and situations first mentioned in clauses, such as *failure* in *Sac failed. Failure occurred during engine start* [3];
- referents not mentioned explicitly, such as the investigated item in *Investigation revealed adequate lube oil* [12];

Processing for pronominals (i.e., pronouns, *one*-anaphora, and zeroes) selects a referent from the *Focus List* (see Section 5.2 and Figure 3). If no previously mentioned entity is appropriate, the system uses a default. For example, in analyzing *Replaced sac*, the default *ship's force* is taken as the agent (see Section 4.1 for the decomposition of *replace*). The referents for full noun phrases and implicit referents are selected from entities mentioned in the discourse. If there is no explicit referent for a definite noun phrase or for a noun phrase without a determiner, the system associates the entity with one in focus via an implicit associate relationship. Indefinite noun phrases are assumed to introduce new referents. While this is is not strictly correct [1], it seems to be sufficient for the CASREP domain. After a referent is found, control returns to semantics.

5.2 Discourse Integration

Discourse integration occurs following the completion of semantic analysis of each sentence. This stage of processing updates the list of discourse entities with entities first mentioned in the current sentence, and maintains a list of entities that are available as referents of pronouns (the *Focus List*). The output of discourse integration is the *Integrated Discourse Representation* (IDR), which represents the entities and situations which have been mentioned in the discourse, and temporal relationships among the situations. Figure 3 shows a simplified version of the IDR produced by the analysis of the CASREP message shown with its summary in Figure 2.

5.3 Time Analysis

PUNDIT's temporal analysis component distinguishes between potential and actual situations, and determines what kinds of intervals are associated with the latter and when they occur [14,13]. Three temporally distinct types of situations are distinguished: states, processes, and events. Each is represented with a distinct *temporal structure* indicating the time intervals over which the situation holds. The *temporal location* of a situation is represented in terms of

```
FOCUS LIST:  [[replace1],[force4],[compressor8],sac,[reveal1],
              [investigate1],[lose1],[operate2],[pressure3]]

Ids:
id(ships_force,[force4])          id(starting_air_system,[starting_air_system9])
id(sac,[compressor8])             id(sac,[compressor9])
id(state,[replace2])              id(event,[replace1])
id(lube_oil,[oil3])               id(pressure,[pressure3])
id(state,[lose2])                 id(event,[lose1])

Events and Processes:
event(
  [lose1]
  becomeP(loweredP(patient([pressure3])))
    belowP(goal(_22866),ref_pt(_22868))
    belowP(goal(_22866),source(_22877))
  moment([lose1]))

event(
  [replace1]
  causeP(agent([force4]),
         becomeP(exchangedP(object1([compressor8]),object2([compressor9]))))
    becomeP(exchangedP(object1([compressor8]),object2([compressor9])))
    missingP(theme([compressor8]),source([starting_air_system9]))
    includedP(theme([compressor9]),goal([starting_air_system9]))
  moment([replace1]))

States:
state(
  [lose2]
  loweredP(patient([pressure3]))
    belowP(goal(_22866),ref_pt(_22868))
    belowP(goal(_22866),source(_22877))
  period([lose2]))

state(
  [replace2]
  exchangedP(object1([compressor8]),object2([compressor9]))
    missingP(theme([compressor8]),source([starting_air_system9]))
    includedP(theme([compressor9]),goal([starting_air_system9]))
  period([replace2]))

Complete Time Relations:
coincide(moment([lose1]),moment([operate2]))
precedes(moment([lose1]),moment(discourse_time))
precedes(moment([replace1]),moment(discourse_time))
precedes(moment([operate2]),moment(discourse_time))
```

Figure 3. IDR for CASREP message

three temporal indices [15]: a moment associated with the situation interval(s) (Reichenbach's *event time*); the *discourse time*, the time at which the text was produced (Reichenbach's *speech time*); and possibly a third time introduced by the past perfect tense (Reichenbach's *reference time*). PUNDIT currently processes certain relational adverbial phrases and clauses [17], such as the prepositional phrase in *Loss of lube oil pressure during operation*. Determining when the loss and operation occurred involves deciding whether to interpret the fragment as past or present, which in turn depends on the lexical aspect of *loss* [10], and on the interpretation of the prepositional adverb *during*. The result of processing *during* is the relation

```
coincide(moment([lose1]),
         moment([operate2]))
```

which means that some moment of the loss coincided with some moment of the operation. Since the lexical aspect of *loss* indicates past time, we also have the relation

```
precede(moment([lose1]),
        moment(discourse_time))
```

as shown in Figure 3.

In order to determine the temporal structure of a predication, PUNDIT needs two kinds of input: the predicate/argument structure produced by the semantic component, and the surface tense and aspect markings (e.g., perfect or progressive). The predicate/argument structure includes aspectual operators [5], which represent the inherent lexical aspect of the verb as stative (no aspectual operator), active (includes the operator `doP`) or eventive (includes the operator `becomeP`). These aspectual distinctions are relevant for verbs whose arguments are simple concrete entities (e.g., *fail* in *the compressor failed* where the verb and its argument refers to a single *fail* event). Another type of lexical semantic information used in temporal analysis pertains to the distinction between verbs in this class, and verbs whose arguments are themselves events (e.g., *occur* in *a failure occurred*). The temporal component recognizes three such verb classes, making it possible, among other things, to derive similar temporal information for the two different ways of referring to a *failure* event (e.g., *Compressor failure occurred* and *The compressor failed*) [13].

6 Portability

PUNDIT has been carefully designed with portability in mind: For example, the modular design of the system ensures that the domain-independent and domain-specific components of the system are kept completely separate. This modularity greatly facilitates porting PUNDIT to new domains, since such a port requires modification only of the domain-specific knowledge bases. In Figure 1, the rectangular boxes represent the domain-independent modules, and the diamond-shaped boxes represent the domain-specific knowledge bases. We have also implemented a number of modules and tools designed to further enhance PUNDIT's portability, which are described in the remainder of this section.

6.1 Automatic Generation of Selectional Restrictions

We have implemented the interactive module SPQR[4] [9] which presents to the user syntactic co-occurrence patterns to be validated as they are generated during parsing. This interactive program greatly reduces the number of incorrect parses generated as well as the number of grammar rules tried, because any partially constructed parse containing an invalid syntactic pattern fails immediately. More importantly, however, the information about syntactic patterns can be stored, with patterns classified as good or bad, so that no pattern is ever presented to the user more than once. The system can thus interactively acquire domain-specific semantic information, and effectively bootstrap into a new domain.

SPQR operates by presenting to the user a syntactic pattern found in the ISR, such as *subject-verb-object* or *noun-adjective*, and querying him/her about the acceptability of that pattern. When presented with a syntactic pattern, the user can

respond to the query in one of two ways. If the pattern describes a relationship that can be said to hold among domain entities, the user accepts the pattern, thereby classifying it as good. An example of a good pattern is the n/pp pattern [*loss*, *of*, *sac*], generated by the fragment *Loss of second installed sac*. After encountering a good pattern, SPQR continues the analysis of the ISR, and the parsing of the sentence is allowed to continue. If, however, the pattern describes a relationship among domain entities that is not consistent with the user's domain knowledge or with his/her pragmatic knowledge, the user rejects it, thereby classifying it as bad, and signalling an incorrect parse. An example of a bad pattern is the subject-verb-object pattern [*loss*, *install*, *sac*], generated by the semantically anomalous analysis of the sentence *Loss of second installed sac* in which *installed* is taken as the main verb. Encountering a bad pattern causes the restriction which checks selection to fail, and as a result, the parse under construction is immediately failed, and the parser backtracks. As the user classifies these co-occurrence patterns into *good patterns* and *bad patterns*, they are stored in a pattern database which is consulted before any query to the user is made. Thus, once a pattern has been classified as good or bad, the user is not asked to classify it again.

6.2 The Development Environment

PUNDIT's development environment includes a set of tools which simplify writing, modifying and testing code, as well as porting PUNDIT to new domains. These include: a set of global switches to tailor the operation of PUNDIT and the generation of trace messages to the specific module of the system one is working on, a general Prolog structure editor that facilitates traversing and editing Prolog terms, a lexical entry procedure that assists in the task of creating new lexical entries, and a semantic rule editor that assists in the creation of consistent semantic rules.

[4]Selectional Pattern Queries and Responses

6.3 Testing the System

We have implemented a set of procedures to automate the updating and testing of PUNDIT. In order to make revisions to PUNDIT, files containing updated code are deposited in a special directory, and the whole system is re-built. Next, PUNDIT processes the entire corpus of messages from each domain, and the resulting output is automatically verified to ensure consistency with the expected, stable output. Any discrepancies between the expected and actual output are flagged so that changes can be made to the update files. This testing procedure has proved invaluable for maintaining the coverage and integrity of the system, and for guarding against unforseen interaction of multiple changes to the system.

7 Implementation

PUNDIT is currently implemented in Quintus Prolog 2.2, and has been ported to TI-Prolog and C-Prolog. The system can run on any hardware supporting one of these Prologs, and has been demonstrated on Vaxes, Suns, Symbolics, Xeroxes, and Explorers. In a typical domain (CASREPS), the PUNDIT system includes approximately 20,000 lines of code, which make up 2000 predicates, containing 6000 clauses distributed among 80 files. Typical sentence-processing times on a Sun 3/60 are in the range of 5–10 seconds for sentences of average length (approximately 10 words); discourse-processing times for entire messages vary from about 15 seconds for short messages to 45–60 seconds for longer and more complicated texts.

References

[1] Deborah A. Dahl. Determiners, entities, and contexts. In *Proceedings of TINLAP-3*, Las Cruces, NM, January 1987.

[2] Deborah A. Dahl. Focusing and reference resolution in PUNDIT. In *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986.

[3] Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Passonneau. Nominalizations in PUNDIT. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, CA, July 1987.

[4] John Dowding and Lynette Hirschman. Dynamic translation for rule pruning in restriction grammar. In *Proceedings of the 2nd International Workshop On Natural Language Understanding and Logic Programming*, Vancouver, B.C., Canada, 1987.

[5] David Dowty. *Word Meaning and Montague Grammar*. D. Reidel, Dordrecht, 1979.

[6] Lynette Hirschman. Conjunction in meta-restriction grammar. *Journal of Logic Programming*, 4:299–328, 1986.

[7] Lynette Hirschman. A meta-rule treatment for english *Wh*-constructions. In *Proceedings of META88 Workshop on Meta-Programming in Logic Programming*, Bristol, 1988.

[8] Lynette Hirschman and Karl Puder. Restriction grammar: a prolog implementation. In D.H.D. Warren and M. Van-Caneghem, editors, *Logic Programming and its Applications*, pages 244–261, Ablex Publishing Corp., Norwood, N.J., 1986.

[9] François-Michel Lang and Lynette Hirschman. Improved portability and parsing through interactive acquisition of semantic information. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, TX, February 1988.

[10] Marcia C. Linebarger, Deborah A. Dahl, Lynette Hirschman, and Rebecca J. Passonneau. Sentence fragments regular structures. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, NY, June 1988.

[11] Martha Palmer. *Driving Semantics for a Limited Domain*. PhD thesis, University of Edinburgh, 1985.

[12] Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Schiffman] Passonneau, Lynette Hirschman, Marcia Linebarger, and John Dowding. Recovering implicit information. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, August 1986.

[13] Rebecca J. Passonneau. A computational model of the semantics of tense and aspect. *Computational Linguistics*, 14(2):44–60, 1988.

[14] Rebecca J. Passonneau. Situations and intervals. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 16–24, 1987.

[15] Hans Reichenbach. *Elements of Symbolic Logic*. The Free Press, New York, 1947.

[16] Naomi Sager. *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, 1981.

[17] Carlota S. Smith. Semantic and syntactic constraints on temporal interpretation. In P.J. Tedeschi and A. Zaenen, editors, *Tense and Aspect*, Academic Press, New York, 1981.

Hopkins, W., Hirschman, L., and Smith, R. "Or-Parallelism in Natural Language Parsing", to appear in *Parallel Algorithms for Machine Intelligence and Pattern Recognition*, eds., V. Kuman, P.S. Gopalkrishman, and L. Kanal, Springer-Verlag, New York, 1989.

OR-PARALLELISM IN NATURAL LANGUAGE PARSING[1]

William C. Hopkins
Lynette Hirschman
Robert C. Smith

Paoli Research Center
Unisys Defense Systems
P.O. Box 517
Paoli, PA 19301

ABSTRACT

We report on a series of simulation experiments for a large-scale natural language processing system. The results indicate that an or-parallel, all-solutions search provides substantial speed-up (20-30 fold) for this application. Longer sentences show relatively greater speed-up, with parse times that increase ***linearly*** with sentence length, given a sufficent number of processors. These results have been obtained using a simple, application-specific model of independent, non-communicating or-parallel processes in a shared memory environment. Simulations run with a range of overhead costs show significant benefits from parallel processing with per-process overheads ranging as high as the median process size; our estimates of overhead times are substantially smaller than median process size.

---

1. This is a revised and expanded version of Hirschman, Hopkins, and Smith, "Or-Parallel Speed-Up in Natural Language Processing: A Case Study," published in ***Logic Programming: Proceedings of the Fifth International Conference and Symposium***, MIT Press, 1988.

This research has been sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Rome Air Development Center (RADC) of the Air Force Systems Command under contract F30602-86-C-0093 and in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research. The new work reported here has been supported by Unisys Independent Research and Development funds.

1. INTRODUCTION

We report here on a series of simulation experiments for a large-scale natural language processing system; our goal was to determine whether processing times in a large-scale application could be reduced substantially by selective exploitation of or-parallelism in a shared-memory environment. We were particularly interested in the granularity of process duration relative to overhead and in the distribution of opportunities for parallelism. We chose a simple, application-specific model of an all-solutions or-parallel search implemented as independent, non-communicating processes in a shared memory environment.

The simulation results show substantial speed-up for this application: at 50% processor effectiveness (speed-up factor divided by number of processors), the speed-ups are typically in the 20-30 fold range. The longer the sentence, the greater the speed-up, giving parallel parse times that increase ***linearly*** with sentence length if a sufficient number of processors are available. Simulations were run with a range of process initiation overhead costs; they show that significant benefits are obtained from parallel processing with overheads ranging as high as the median process duration, although our estimates of overhead times are substantially smaller than that. Pruning the search space through semantic filtering of partial parses decreased the available parallelism, as expected, but (surprisingly) also decreased the time to termination of the overall search, due to the pruning of lengthy, but unsuccessful, search paths.

We chose our application (a top-down parser with a broad-coverage English grammar) because it represented a large-scale logic program written in Prolog, developed independently of any notion of parallel processing. We limited our initial experiments to one particular type of application level or-parallelism, parallel exploration of disjunction (alternatives) in grammar rules, because this level of parallelism turned out to produce substantial exploitable parallelism. It is clear that there are *many* other opportunities for parallelism in the application, but these were not explored.

2. BACKGROUND

There have been a number of approaches for the implementation of or-parallelism, including the "Gigalips" project collaboration at Argonne Labs, the University of Manchester and the Swedish Institute of Computer Science [Ciepielewski1985, Disz1987, Shen1987, Warren1987, Hausman1987], work at various institutions in Japan [Moto-oka1984, Sohma1985], and research at a number of other sites [Conery1987, Westphal1987]. Or-parallelism has also been addressed within the model of Concurrent Prolog [Shapiro1987]. One goal of these

researchers has been to support a transparent port of an application to a parallel Prolog (possibly with minimal annotation to indicate which goals should be run in parallel).

The focus of our research has been on demonstrating that large-scale applications exhibit parallelism with useful granularity, rather than on the underlying or-parallel implementation. We therefore adopted a simple model of independent, non-communicating or-parallel processes, to minimize issues such as shared variable bindings. This model resembles the PIE model [Moto-oka1984], and has some similarities with Shapiro's all-solutions approach in Flat Concurrent Prolog[2]. The limited model of parallelism outlined here would also be appropriate for application of the Kabu-Wake method [Sohma1985]. The grain size we found, however, differs from most of these approaches since we focus on an application-level disjunction for spawning or-processes, rather than the finer-grained Prolog disjunction. Our work also differs from many of these [Sohma1985, Hausman1987], in that process creation is determined solely by the structure of the program, not by any run-time considerations. Dynamic splitting of processes on demand requires communication and structural overheads that are not required in our simpler model. The compensating cost of our approach is that it must handle more processes than necessary to keep the system busy.

One aspect of our application deserves mention, namely parsing to all solutions. Support for the all-solution approach to parsing becomes particularly important when applying broad-coverage grammars to speech understanding, where there is substantial indeterminacy in the input signal; the size of the resulting search space for parsing will necessitate massive parallelism to explore it efficiently. The importance of the speech processing application alone would justify special application-specific architectures tailored for parallel parsing, although use of an application-independent model of parallelism would clearly be both more general and more flexible.

Our all-solutions application explains why our results show substantial parallelism, even with an overhead cost comparable in size to average process duration; it may also explain why our results differ from the negative results reported by Fagin and Despain [Fagin1987] who looked only at single solution search in most cases and, not surprisingly, found little speed-up due to or-parallelism.

---

2. In an all-solutions environment, or-parallelism merges with and-parallelism, because exploration of alternatives becomes mandatory: alternatives become indistinguishable from independent, non-communicating and-processes.

3. DESCRIPTION OF THE APPLICATION

Our experiments are based on the parser and grammar of the Unisys PUNDIT[3] natural language processing system [Hirschman1989]. The PUNDIT system is a broad-coverage natural language processing system, designed to convert information contained in natural language text (e.g, messages or reports) into database or knowledge base entries; the same system can also convert natural language queries into database queries, to provide natural language access to databases/knowledge bases. The full PUNDIT system consists of separate modules for processing syntactic, semantic and pragmatic information. The full system currently consists of some 30,000 lines of Prolog code. It runs in Quintus Prolog under Unix (Sun, Vax) and in TI Prolog on the Explorer Lisp Machine. The syntax module, the focus of our experiments in this paper, makes up approximately one third of the system (10,000 lines of code); it includes the parser (1000 lines), grammar (3000 lines), syntactic regularization component (500 lines), and syntactic-based selection component (5000 lines). The lexicon is an additional 4000 lines and contains 2000 lexical entries.

3.1. Restriction Grammar

For purposes of these experiments on parallelism, we focused only on the syntactic module of PUNDIT, namely Restriction Grammar [Hirschman1985]. Restriction Grammar is a form of logic grammar, which consists of context-free BNF definitions, augmented with constraints or ***restrictions***. However, unlike Definite Clause Grammar [Pereira1980] or its variants, there are no explicit parameters in Restriction Grammar. Table 1 shows a (highly simplified) Restriction Grammar; this grammar illustrates inclusion of restrictions (indicated in standard logic grammar form by {}), as well as both explicit disjunction within grammar rules (indicated by the connective ";") and implicit disjunction (indicated via multiple definitions); compare the rules for ***article*** and ***predicate***. The grammar reader recognizes both forms of disjunction and treats them equivalently.

The English grammar consists of approximately 125 BNF definitions and 55 restrictions. This base grammar is augmented by a meta-rule which generates additional rules for handling conjunction, increasing the size of the grammar by about 50%, to approximately 180 definitions. The coverage of the grammar is very broad; it includes an extensive treatment of co-ordinate conjunction [Hirschman1986], detailed coverage of complex subject and complement types, fine-grained analysis of noun phrases, treatment of relative clauses and questions [Hirschman1988], and also a small set of definitions for

---

3. *Prolog UNDerstanding of Integrated Text.*

Sample Grammar		
sentence	::=	subject, predicate, {subj_verb_agree}.
subject	::=	noun_phrase.
noun_phrase	::=	left_mods, *noun, right_mods.
left_mods	::=	article, adjectives.
right_mods	::=	prepositional_phrase; null.
article	::=	*determiner; null.
adjectives	::=	*adjective, adjectives.
adjectives	::=	null.
predicate	::=	*verb, object.
predicate	::=	*verb, {be_verb}, be_predicate.
object	::=	({transitive_verb}, noun_phrase); ({intransitive_verb}, null).
be_predicate	::=	*adjective; prepositional_phrase.
prepositional_phrase	::=	*preposition, noun_phrase.
null	::=	~ .

Key:	
;	Disjunction
,	Conjunction
~	Empty Element
*	Terminal Grammatical Category
{...}	Restriction

Table 1: Example of a Simple Restriction Grammar.

handling fragmentary sentences (e.g., ***Metal particles in oil*** or ***Suspect coupling to be sheared***) [Linebarger1988].

The result of the broad coverage of the grammar is that parsing involves a good deal of search. In this corpus, the ratio of definitions tried vs. number of nodes in the tree is approximately 25 [Lang1988]. There are approximately 350 disjunctions in the grammar (an average branching factor of about 2). Many sentences receive multiple parses; even sentences receiving only a single parse still explore many alternative paths that end in failure.

The grammar in Table 1 is highly simplified, but shows several disjunctive definitions, both explicit (***right-mods, article, object, be_predicate***), and implicit (***adjectives, predicate***). Disjunction in grammar rules (both explicit and implicit) is the level at which we

spawn parallel or-processes[4]. Note that in this small grammar, the second disjunct in many cases is the empty (*null*) definition. It would seem that exploring an empty definition in parallel with a non-empty definition would not yield significant speed-up. However, the parse trees diverge upon acceptance of an empty definition vs. a non-empty one, because the words left in the word-stream are then different. This can lead to very significant backtracking in a sequential system. Thus null definitions turn out to be an extremely significant source of parallelism.

3.2. The Mechanism of Restriction Grammar

The parser operates in a top-down left-to-right manner, and constructs the parse tree automatically as it invokes rules. The nodes of the parse tree are simply the left-hand side of each BNF definition used by the parser in its current search or proof. At the leaves of the completed parse tree are terminal grammatical categories (e.g., ***determiner, noun, verb***), associated with the words from the sentence being parsed. A partial parse tree is shown in Figure 1, using the simplified grammar from Table 1.

Restrictions capture context-dependent constraints as well-formedness constraints on the parse tree. However, constraints only ***filter***, they never ***add*** information to the parse tree. If a constraint *succeeds, parsing continues;* it is fails, the parser backtracks and looks for alternative definitions to apply. Restrictions are implemented using a limited set of operators which traverse and inspect the current state of the parse tree. These operators include relations for ***label*** (used to check the name or label of a node), ***parent***, ***child***, ***left/right_sib***, and ***word***.

3.3. Restriction Grammar Data Structures

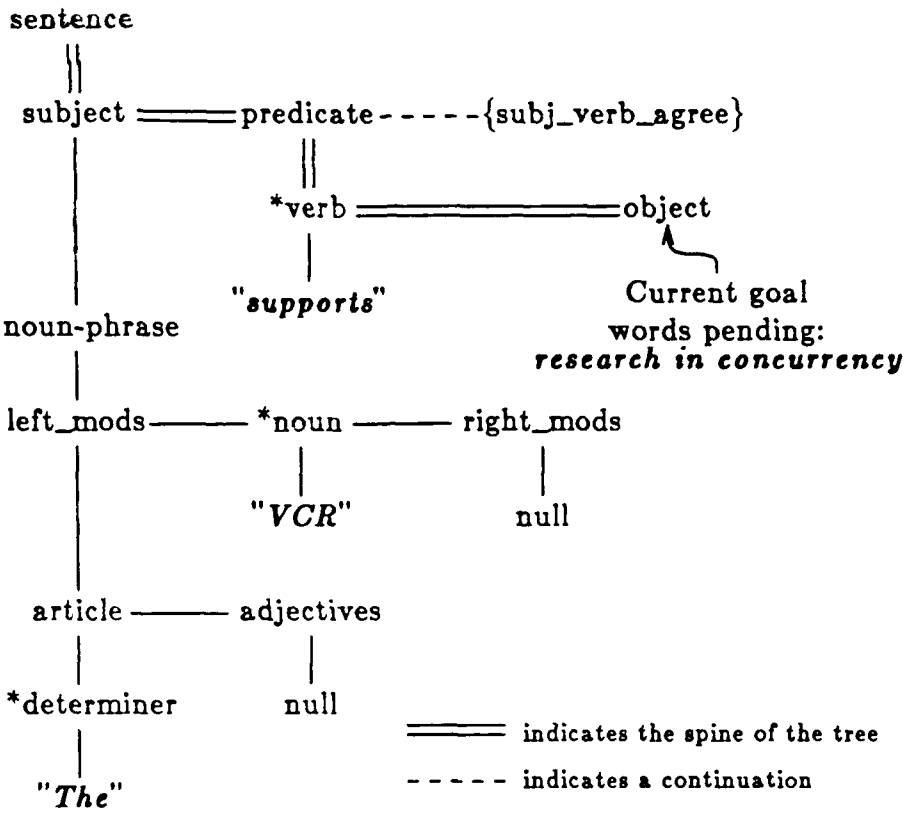
To support automatic tree building and unrestricted tree traversal, the parser maintains the parse tree as a data structure composed of `link`, `treeTerm`, and `path` structures. This structure separates the forward pointers (contained in the `TreeTerm` structure, used for traversing down the parse tree, or to the right) from the backward pointers (contained in the `Path` structure, used for traversing up or to the left):

```
link(TreeTerm,Path)
```

Maintaining this data structure makes grammar rule execution more complex (and thus larger grained) than a comparable operation in a plain DCG.

---

4. There are many additional opportunities for or-parallelism, as well as for independent and-parallelism that we have not exploited in this simulation; this will be the subject of future work.



The VCR supports research in concurrency

Figure 1: Partial Parse Tree.

The tree-term data structure contains fields for the node name, the first child, the right-sibling, the word (unfilled, except for terminal grammatical categories), and the regularized syntactic representation, which is constructed incrementally during the parse:

```
treeTerm(NodeName,FirstChild,RightSib,WordField,RegSyn)
```

The child and sibling fields are themselves tree-terms. Note that the fields of the tree-term correspond to the operations ***label, child, right_sib*** and ***word*** used in the restrictions.

The second argument of the `link` data structure is the path back to the root, which is also a recursive data structure, consisting of a direction functor (`up` or `left`) with two arguments, namely the tree-term reached by moving up or left, and the path from that point back to the top:

```
Path = up(ParentTreeTerm,PathToTop)  or
Path = left(LeftSibTreeTerm,PathToTop)
```

Note that the path structure supports the restriction operators of ***parent*** and ***left_sib.***

3.4. The Restriction Grammar Interpreter

The Restriction Grammar parser is implemented by an interpreter (instrumented for debugging) and a dynamic translator (for parsing with a stable grammar) [Dowding1987]. The top-level routine of the interpreter, exec, has a clause for each type of connective that can be found in a grammar rule: conjunction, disjunction, terminal nodes (indicated by an asterisk), literals, and restrictions (indicated by curly braces). A few simplified clauses of the interpreter are shown in Figure 2. The exec predicate has five arguments: the BNF definition being interpreted, a pair of link structures, First and Last, which identify the current site for tree-building on procedure entry and procedure exit, and a pair of parameters for the word-stream on procedure entry and on procedure exit. The top-level call is

```
%%% Conjunction
exec((X,Y),First,Last,InWords,OutWords):-!,
    exec(X,First,Next,InWords,WordsAfterX),
    exec(Y,Next,Last,WordsAfterX,OutWords).
%%% Disjunction
exec((X;Y),First,Last,InWords,OutWords):-!,
    (exec(X,First,Last,InWords,OutWords);
    (!,exec(Y,First,Last,InWords,OutWords))).
%%% Terminal Node
exec(*X,First,Last,InWords,OutWords) :-!,
    label(First,X),              % label the node "X"
    attach_word(X,First,InWords,OutWords),
    right_sib(First,Last).       % build next sib
%%% Nonterminal
exec(X,First,Next,InWords,OutWords):-!,
    label(First,X),              % label the node "X"
    X ::= Y,                     % pick up BNF defn
    child(First,Child),          % create 1st child node
    exec(Y,Child,_Sib,InWord,OutWords),
       right_sib(First,Next). % create next sib node
```

Figure 2: exec **Predicate from Grammar Interpreter**

```
exec(sentence,InitialNode,LastNode,SentenceWords,[]).
```

where InitialNode and LastNode are link structures and SentenceWords contains the words in the word string to be parsed. It is important to remember that the BNF is purely data, and that the connectives (comma and semicolon) are the (uninterpreted) functors used to build the grammar data structure.

3.5. The Continuation-Based Interpreter

We chose to use the interpreter version of the Restriction Grammar as the basis for the simulation experiments, because it supports the instrumentation required to generate input to the simulator. In order to simplify the execution model, we modified the grammar interpreter to use a continuation-based interpreter. At each grammar rule, the continuation-based interpreter constructs an explicit continuation to be passed to the new process. The continuation contains the pending conjoined goals and makes each process self-contained, with no dependence on the Prolog stack. A simplified fragment of the continuation-based interpreter is shown in Figure 3 for comparison.

The continuation-based interpreter has four parameters: the rule being interpreted, the node being built, the input word stream and the continuation stack. When a conjoined rule is encountered, the first conjunct is executed and the second is put on the continuation stack. Use of this stack means that the interpreter no longer needs to pass around paired parameters for tree-building or for the word-stream. It simply calls exec iteratively, updating the word stream and tree-building site with each call. The initial call is

```
exec(sentence, InitialNode, SentenceWords, []).
```

When it completes execution, it pops the stack to get more work, until the stack is empty, at which point it is done. When the interpreter pops the stack, it begins building new nodes wherever it finds itself in the tree. The rule for non-terminal expansion (invoked for the last conjunct in a definition) pushes a special marker onto the stack, marking the last child as having no right sibs. When the interpreter's pop procedure finds this on the *top* of the stack, it removes it, climbs to the parent to resume tree building, and pops the stack again, looking for more work.

In the continuation-based interpreter, each disjunct can be thought of as a self-contained process which runs to completion (or failure) completely independently of other processes. All that is required for a process spawn is a "clean copy" of the process state (given by the current tree node, word stream, and continuation stack), with logical variables in this structure named apart from logical variables in other processes. This removes any dependency on the Prolog stack, making it possible to

```
%%% Conjunction: push continuation onto the stack
exec((X,Y),First,InWords,Stack):-!,
    exec(X,First,InWords,stack(Y,Stack)).
%%% Disjunction
exec((X;Y),First,InWords,Stack):-!,
    (exec(X,First,InWords,Stack);
      (!,exec(Y,First,InWords,Stack))).
%%% Terminal expansion
exec(*X,First,InWords,Stack):-!,
    label(First,X),           % label current node "X"
    attach_word(X,First,InWords,OutWords),
                  % attach word & remove from stream
    right_sib(First,Next),
                  % get sib, to resume tree building
    pop(Next,OutWords,Stack).% pop stack
%%% Nonterminal expansion
exec(X,First,InWords,Stack):-!,
    label(First,X),           % label new node "X"
    X ::= Y,                  % get new BNF defn
    child(First,Child),       % build first child
    exec(Y,Child,_Sib,InWords,stack(no_sibs,Stack)).
                              % stack no-sib marker

%%% pop
pop(_Node,[],~) :- !.         % stack empty, finished.
% non-terminal case: pop node off stack,
%    move up to parent node and pop stack again,
%    until a real conjunction is reached.
pop(Node,Wds,stack(no_sibs,Stack)):- !,
    up(Node,Parent),          % no sibs, find parent
    right_sib(Parent,Next),   % get next sib
    pop(Next,Wds,Stack).      % pop stack for more work
% conjunction case: pop node and continue
pop(Node,Wds,stack(Y,Stack)) :- !,
    exec(Y,Node,Wds,Stack).
```

Figure 8: Continuation-based exec **Procedure**

run crude timing experiments on the state copying operation. The exact structures that were copied as well as those that could be shared are described in greater detail in section 5.

4. APPLICATION BEHAVIOR MODELS

The experimental approach uses a model-based simulation system named the ***Virtual Computation Recorder*** or ***VCR*** [Hopkins1987], which infers the concurrent behavior of the parser from its sequential behavior. Two processing models for the PUNDIT parser drive the simulation. The ***sequential execution model*** is derived from sequential Prolog execution, and defines the granularity of the simulations. The ***concurrent execution model*** is a special case of a shared-memory or-parallel execution model for Prolog. These models and their relationship are a key element of the research approach.

4.1. Sequential Model

The PUNDIT parser searches for all parses of the input string, using standard Prolog backtracking to find all parses. A single parse tree is built and unbuilt in backtracking, so there is only one partial parse tree, representing the state of the currently active parse, at any time. Nodes in the tree represent the symbols in the grammar; for any node representing a symbol, its children represent the (conjoined) symbols in the definition used to expand the symbol. Disjunction in the grammar is not represented in the tree, since only the choice actually used is represented.

A short description of how the parse tree evolves may prove useful here. Figure 1 above shows the state of the parse tree at the time that the definition for **object** is to be interpreted. The first rule for **object** is successful, building nodes for **object**, **noun_phrase**, **left_mods**, etc. to match the phrase ***research in concurrency***. After the successful parse is reported, the alternatives to the rules applied in generating the **object** subtree are tried in the usual depth-first backtracking order. Specifically, starting with the last (rightmost) goal to succeed, the remaining disjuncts are applied until all fail, causing backtracking to the next goal to the left (or, if there is none, to the parent) and retrying it. The nodes created by a rule are removed from the tree when the rule's alternatives are tried. Eventually, when the **noun_phrase** goal in the first **object** rule is retried and fails, the parse tree is restored to the state in Figure 1, the **object** rule is retried, and the second **object** rule (**object ::= null**) is applied. (The restrictions in the **object** rule do not alter the tree, and have been omitted from this discussion.)

4.2. Concurrent Model

The concurrent execution model of the parser supports the concurrent construction of independent parse trees, spawning processes in shared memory for the disjunctive goals encountered. Each child process will attempt to complete the partial parse supplied by its parent

using one of the disjoined rules. Spawning is under the control of a policy which specifies which disjuncts are spawned and which are left to be pursued with the normal failure/backtrack mechanisms. At one policy extreme (never spawning), the model is that of sequential Prolog; at the other (always spawning), no backtracking occurs, as each disjunct is explored by its own process. Intermediate policies that we considered included conditioning the spawning statically on the grammar rule, or dynamically on the current resource availability. The spawning policy is applied (recursively) to the child processes, so that even a modest branching factor can lead to a large number of processes.

Failure to satisfy a spawned goal results in termination of the process. The cost of backtracking, and of maintaining choicepoint information to control it, can thus be avoided for spawned goals. Each disjunct represents a distinct attempted parse, and has its own parse tree, which is initially a copy of the parse tree of its parent. The result of a concurrent parse is the forest of parse trees produced by the successful parses. For simplicity, the rest of the discussion will discuss only spawned disjuncts. Any unspawned disjuncts are treated with the normal Prolog mechanisms.

Referring back to the parse tree in Figure 1, the state shown becomes the basis for two concurrent parses, one for each **object** rule. For goals with more rules (which are common in the actual grammar), there will be a correspondingly larger branching factor.

4.3. Process State

A child process receives the initial state specified by the four arguments of **exec**:

- the BNF rule currently being expanded
- the link structure, which identifies the tree term (parse subtree) associated with the current node, and the return path to the root of the parse tree
- the remaining input string
- the continuation stack, specifying the pending conjuncts in the BNF rule being expanded at each level of the tree

No other data from the parent process is needed. In particular, the parent's Prolog stack contains no data that needs to be communicated, and the child process can initialize its stack independently.

In the concurrent execution model, parent and child processes share those parts of this data which contain no unbound logical variables when the child is spawned. Those portions that contain unbound logic variables must be copied, as summarized in Table 2.

State component	Shared	Local
BNF rule	rule	none
tree term	subtrees *left* of spine	spine
return path	none	return path
remaining words	wordstream	none
continuation stack	continuation stack	none

Table 2: Process State.

As the parse proceeds, the tree is expanded along its right edge. All unbound logic variables in the tree are on the spine (the `tree-term` nodes from the root to the current site of tree-building); the completed subtrees to the left of the spine are fully instantiated, and thus may be shared. The return path is the inverse of the spine, containing back pointers (`up` and `left` structures) along the path to the root The parse tree (Figure 1) for the simplified grammar example identifies the spine and return path with doubled links. The spine itself must be copied, naming apart the logical variables in it for the child process; a new return path corresponds to the new spine. The continuation stack contains no logical variables, only BNF definitions, and may be shared. It is implemented as a recursive data structure `stack`. As tree building proceeds, processes push and pop continuation elements on the stack. Pushing appends the existing stack to a new top element; pop returns the elements after the current top. In each case, processes continue to share the common segments of the data structure, and no copying is needed.

To determine the cost of performing the necessary copying and naming apart of variables, the spine of the tree and the return path were copied explicitly in Prolog. The spine grows and shrinks during the course of parsing and exact copying time depends on the actual length of the spine when copied. We measured the copying times during an actual parse, finding the average copying time to be about 1 millisecond. This corresponds reasonably well with intuition (approximately 50-100 nodes copied, with one logical inference required per node copied on a 50K Lips Prolog). The copying time increased only slightly with the length of the spine, indicating that other costs were being included; we have, however, taken the conservative path of using the measured copying times. The time to start up the spawned process is arbitrarily set at 1 millisecond to allow for process creation and system overheads

These timings all assume a simple model of parallel Prolog in which the copying is not supported as a primitive. It is possible, of course, to migrate the copying process to a lower level of the system implementation, to C or assembler, to make it faster. To keep our results as

general as possible, we have not done so in these experiments.

5. SIMULATION METHODOLOGY

This section describes the Virtual Computation Recorder simulation methodology used in this study. Overloading the abbreviation, ***VCR***, is intentional, with video-cassette technology as a metaphor for the simulation technique. Intuitively, the VCR records program behavior and plays it back on a simulated multiprocessor system, with user control over system characteristics and parameters. Key concepts in the approach are the behavior model and its implications, and abstract interpretation. The previous section defines the sequential and concurrent behavioral models for the PUNDIT parser; we now discuss how these models shape the VCR simulations.

The VCR is intended to support application-based experimentation in concurrency policies and architectural requirements without committing the research ***a priori*** either to a specific policy or to the expense of a real parallel implementation effort. Through such simulation experiments, the researcher can select an implementation strategy based on reliable information about the effectiveness of the available choices.

The specific objective of the VCR is to provide rapid implementation of a simulation system that makes only minimal commitments to any specific choice of model or policy. One key to achieving this objective is the early identification of the degrees of freedom that are to be explored, and those that are fixed; the simulation system, to be effective, must be non-committal on the former. Subsidiary objectives include ease of use, ability to specify experimental measurements, and efficiency of operation.

The VCR meets these objectives, allowing the simulation of concurrent implementations of applications without requiring that the concurrent system be implemented, or in some cases, fully designed. Instead, a model of the parallel system architecture is used to derive an application's concurrent behavior from the sequential behavior of its computation.

The VCR concept is built around a strong, user-specified model of the problem to be studied. The model identifies the constituents of the computation, their necessary ordering, and fixed aspects of the concurrency policies. We call the constituents ***atoms***, to capture the idea of their indivisibility within the model. Based on the model, the existing applications are instrumented to identify the atoms, and the behavior of the computation is recorded in terms of the atoms. This recording process is a form of ***behavioral abstraction*** [Bates1983], producing an abstract ***behavior description*** which becomes the basis for simulation.

The behavior description is not a representation of the program logic, but a record of significant aspects of its behavior for a particular set of inputs. All of the run-time tests, branching and other types of processing that the program may perform are undifferentiated and anonymous unless the model and atom set specifically identify them. Interpretation or ***playback*** of the behavior can therefore be made extremely simple and efficient. Only those aspects of the behavior that are relevant to the research questions at hand are addressed if the model and atom set are properly designed.

Instantiation of the unbound aspects of the model is accomplished through abstract interpretation of the behavior description, creating a new, more specific behavior description. Several varieties of abstract interpreters are possible; we define a ***playback*** device as one that instantiates the time-of-occurrence attribute of the atoms in the behavior description.

5.1. Atoms and Recording

An initial set of atoms is defined in the sequential computation through annotations in the parser program, reflecting the parts of the computation identified by the sequential and concurrent execution models. The models are not concerned with the details of satisfying a goal beyond identifying subgoals and spawning processes for them. Thus, only four types of processing are identified in the sequential execution of the PUNDIT parser: ***start-goal***, which sets a goal, ***complete-goal***, the successful derivation of the goal, ***fail-goal***, the failure to derive the goal, and ***backtrack***, the search back to the most recent alternative goal. The annotations in the grammar interpreter identify the transitions from one type of processing to the next in the course of finding all parses for a given sentence. The annotations thus define four atoms for each symbol in the grammar.

The annotations are used both to generate the behavior description and, separately, to measure the time durations of the atoms. During recording, each annotation in the interpreter invokes a procedure that writes the atom identifier to a file, along with subsidiary information such as the depth in the parse tree and the rule number being expanded. In addition, instances of disjunctions are labeled dynamically, and the labels are propagated through the interpreter to allow the disjuncts to be reassociated later; the interpreter inserts atoms carrying the disjunction label at these points. A single sequential behavior description suffices for all experimentation with a given sentence.

Generation of the atom durations uses the same annotations with different expansions which interface to an execution time profiler. The profiler is similar in spirit to the Unix monitor and profile capability but charges time to the previous annotation encountered, rather than the current procedure. This expansion of the annotations is quite efficient,

causing only a small decrease in performance relative to the unannotated parser; the resulting duration measurements are thus fairly reliable when adjusted for this systematic error in measurement.

5.2. Behavior Description and its Representation

Given a model and a computation, a behavior description is the set of atoms found in the computation and the ordering that the computation imposes. The sequential computation of the PUNDIT parser generates a trace of the atoms in the program execution, imposing a complete ordering on them; a true concurrent parser's computation would generate concurrent traces, imposing a partial ordering.

Various representations of the behavior are possible, with varying amounts of information. A sequential trace, for instance, captures the full linear ordering of a sequential execution of a computation, but does not admit any concurrency unless it includes explicit spawn atoms that specify labeled entry points in the trace. The VCR also uses an alternative form, named an ***Abstract Behavior Graph*** or ***ABG***, to represent a less restrictive partial ordering. An ABG is a directed graph with vertices representing atoms and edges designating successors; spawn atoms represent explicit concurrency in the behavior. Other representations are, of course, possible. The choice of representation depends on what information is of interest and the kind of processing to be performed on it. The VCR currently uses several forms of trace as well as the ABG, according to the problem and questions being addressed.

5.3. Abstract Interpretation

Abstract interpretation is the process of assigning meaning to atoms and computations. The atoms themselves represent pieces of the computation; their interpretation by the VCR is non-standard in the sense that it ignores the original intent of the computation, which is to produce a parse tree for the input sentence. Instead, the interpretation assigns attributes to the atoms which are of interest in the experimental program, such as time of occurrence, process number, processor allocation, and so on. Attributes are also thus derived for the computation, such as time to completion, concurrency, and processor utilization. In some cases, the interpretation may introduce new atoms, representing, for instance, purely concurrent aspects of the behavior.

5.4. Reinterpretations

Most VCR interpretations of a behavior description produce another behavior description. The new description may be in the same form and representation as the previous one, or it may be entirely different. For instance, the concurrent playback interpretation may, at the user's option, generate only a summary description, or it may generate a detailed interleaved trace of the the atoms interpreted. The

term ***reinterpretation*** (more accurately, ***reinterpretable interpretation***) denotes the process of generating an interpretable behavior description. Reinterpretations include building the ABG, transforming it back to a sequential trace, transforming it to a modified set of atoms, and instantiating its attributes.

The sequential recording of the behavior of a parsing run is transformed into a concurrent ABG by two reinterpretations. The first implements the concurrency policy, collecting the disjunction label information and inserting atoms to spawn the individual disjuncts where the disjunction is first encountered. Each spawn atom is tied to a specific part of the search, and the total number of processes spawned in the concurrent parse of the sentence can be determined statically from the resulting behavior description.

The second reinterpretation deletes the ***backtrack*** and ***retry*** atoms that are unneeded in the concurrent system. With all disjunctions being explored concurrently, all such atoms are deleted, so this is a very simple reinterpretation. In the case where not all disjunctions are concurrent, the first reinterpretation would implement the spawning policy, generating information to guide the second.

5.5. Playback

The playback process, which interprets the concurrent ABG to instantiate the time of occurrence of each atom, requires specification of the architectural and process management parameters of the system, and it is at this point that the cases proliferate. Each run of the playback takes only a few minutes to set up and execute, limited mainly by the time to write the detailed trace to disk. Where a single measurement is desired from a number of runs, as for Figure 6, the detailed trace is unnecessary, and each run can be done in seconds.

5.6. Data Reduction and Analysis

The trace of the concurrent playback is a record of each atom's interpretation, with process number, processor number, time of occurrence, etc. derived from the playback process. This data is easily reduced to determine the types of results shown in Section 6 with simple utility programs written in whatever language is convenient; we have used the Unix utilities ***awk***, ***grep***, ***sort***, ***plot***, and shell scripts to generate these results. As an example, the concurrency curve in Figure 4 uses ***grep*** to extract the start-process and end-process atoms, ***sort*** to order them by time, ***awk*** to associate an active process count with each time, and ***plot*** to present the data; a shell script controls the overall process.

Test Sentences	
1.1.1	Starting air regulating valve failed.
4.1.1	While diesel was operating with SAC disengaged, the SAC lo alarm sounded.
4.1.3	Pump will not turn when engine jacks over.
5.1.2	Disengaged immediately after alarm.
6.1.2	Inspection of lo filter revealed metal particles.
9.1.1	SAC received high usage during two BECCE periods.
9.1.4	Loud noises were coming from the drive end during coast down.
22.1.1	Loss of lube oil pressure during operation.
25.1.3	Suspect faulty high speed rotating assembly.
28.1.1	Unit has excessive wear on inlet impellor assembly and shows high usage of oil.
31.1.3	Erosion of impellor blade tip is evident.
31.1.4	Compressor wheel inducer leading edge broken.

Table 3: Maintenance Domain Sentences Used.

6. ANALYSIS OF RESULTS

We performed a large number of experiments, varying the sentence being parsed, the number of processors available, the times for processing atoms, and most important, the overhead associated with spawning a new process. The sentences, listed in Table 3, are drawn from the Navy CASREPS message domain, dealing with equipment casualty reports. (A SAC is a starting air compressor.) This discussion will use the results for sentence 28.1.1[5], ***Unit has excessive wear on inlet impeller assembly and shows high usage of oil.*** The results presented assume that all disjuncts in every BNF definition are explored by separate processes.

6.1. Concurrency and Granularity

Figure 4 shows the concurrency discovered in the parsing of the example sentence; the graph shows the number of active processes during the course of the simulated parse, assuming enough processors (116 in this case) that one is available to run each process when it is spawned. As is evident in the example, the concurrency is irregular, with one or more peaks and potentially long start-up and shut-down tails. Furthermore, the shape of the concurrency varies markedly from

5. The sentence numbers are generated during pre-processing, to provide a unique identifier for each sentence made up of:
message-number.paragraph-number.sentence-number.

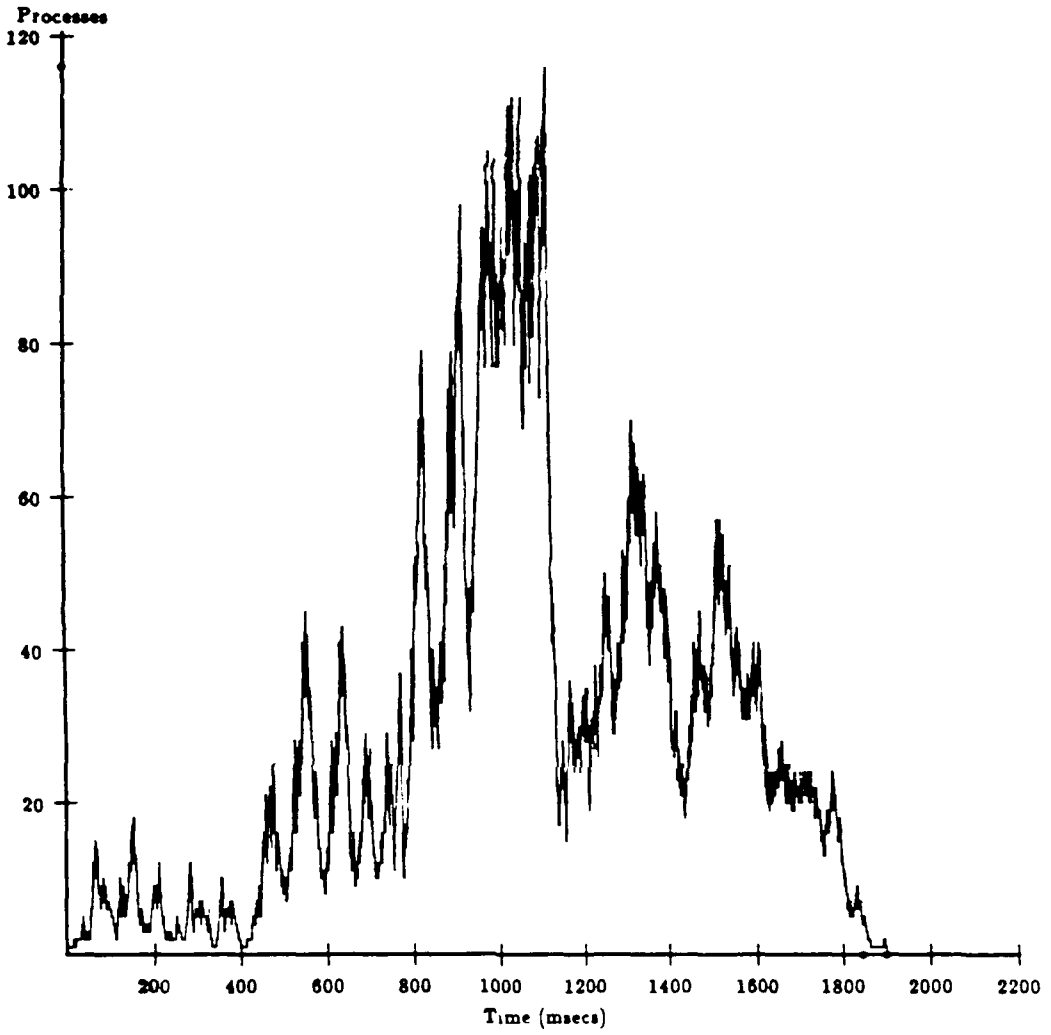


Figure 4: Typical Unconstrained Concurrency

sentence to sentence, depending on where in the sentence multiple parses are attempted and how long the parser follows a path.

Figure 5 shows the active process distribution for the same sentence with a 50 processor limit, which increases the parse time by about 10% from the unlimited case. Processes wait in a strict (FIFO) queue if all processors are busy when they are spawned. The effect of varying the number of processors on speed-up is shown in Figure 6; the curves have a characteristic shape with a distinct knee that reflects the decreasing effectiveness of additional processors. Below the knee, speed-up is close to linear; beyond it, the process queue empties and

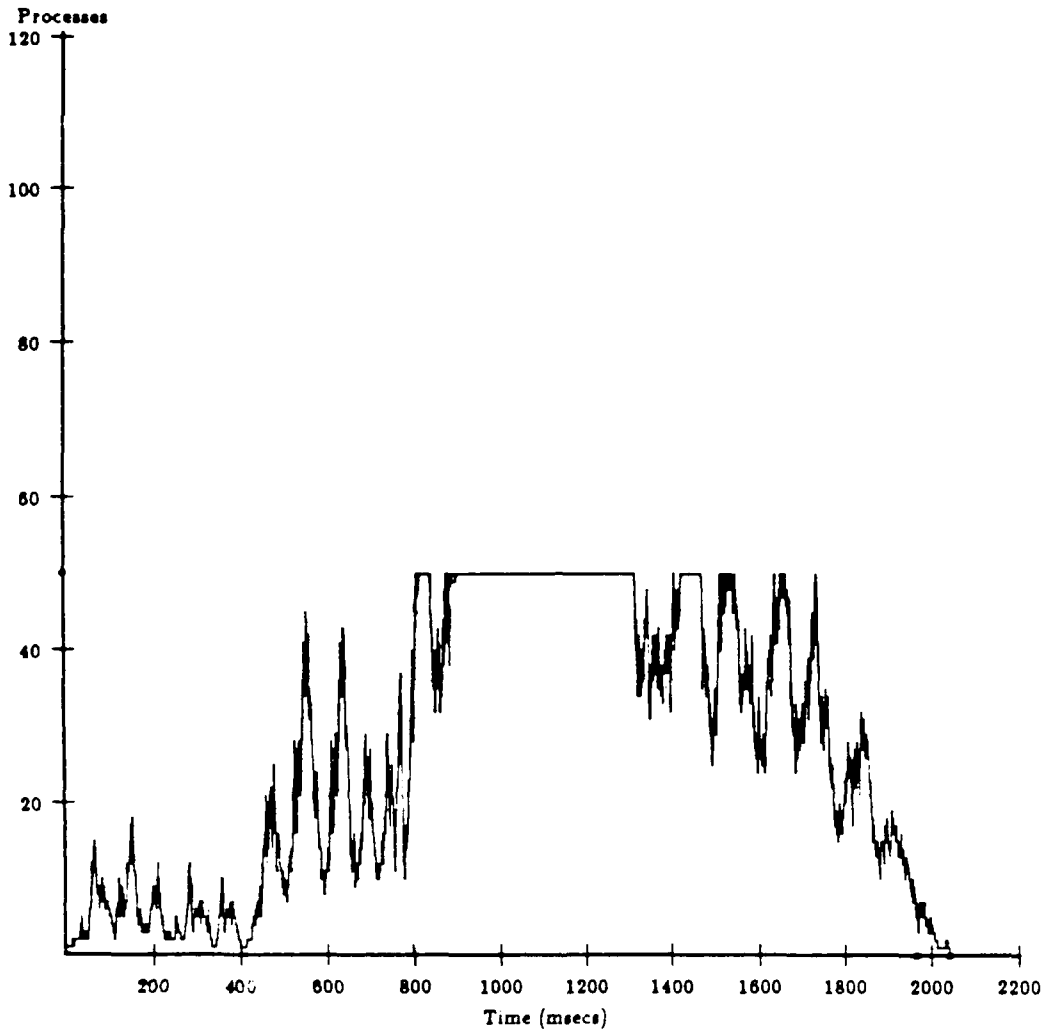


Figure 5: Typical Constrained Concurrency

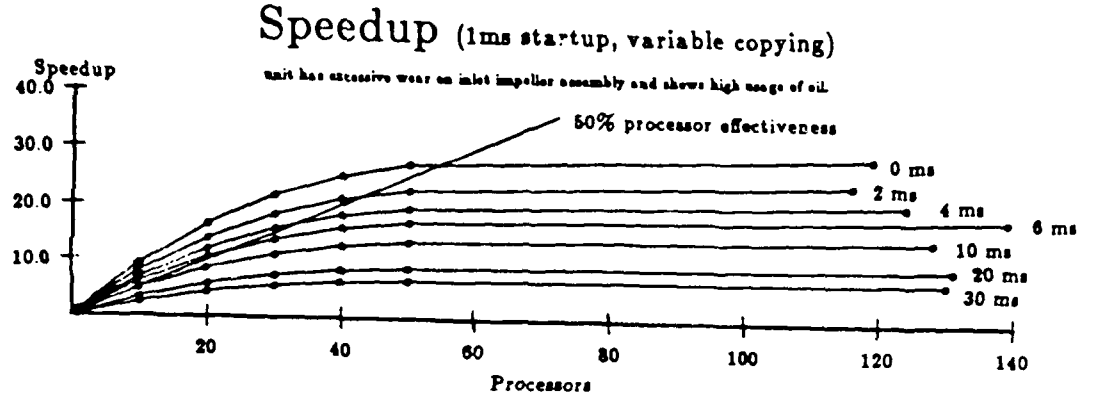


Figure 6: Speed-up Curves with Varying Copying Costs

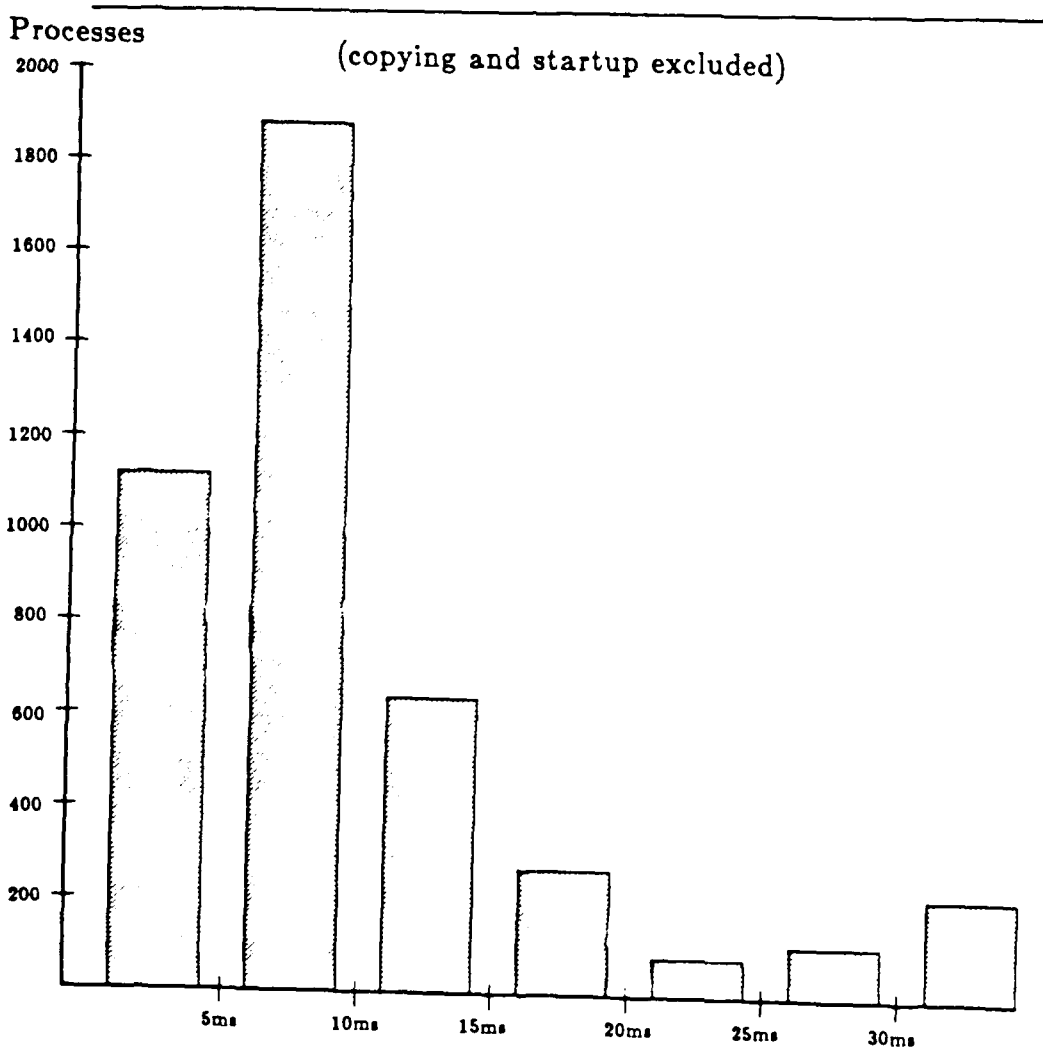


Figure 7: Process Length Distribution

additional processors are increasingly idle.

The duration of the processes and the relative cost of spawning them are extremely significant for matching the processing to a multiprocessor architecture. Our results here are quite encouraging: the median process time, using measured atom timings from interpreted Prolog runs, is typically around 9 milliseconds. This provides a useful ratio of processing time to process start-up and copying time (9 ms compared to 3ms or approximately 3:1; see Sections 6.2 and 6.3 for a discussion of the effect of process overhead on speed-up). Figure 7 shows the distribution of process duration in 5 millisecond increments; the shape of this distribution is remarkably invariant over the sentences tested.

6.2. Speed-up

The calculation of parallel speed-up factors, especially the selection of the base time, is notoriously open to "optimization of results." The generally accepted guideline is to compare to the best sequential implementation, to get a fair measure of system design choices. Contrary to this guideline, we have taken the sequential parse times of the unmodified PUNDIT grammar interpreter parser as our baseline, since the instrumentation technique required its use; an additional doubling of speed is obtainable by translating the grammar directly to PROLOG. Section 9 addresses the implications of this speed-up. The timings are reconstructed from the measured behavior and atom durations of the parser (modified for continuation stacking), normalized for systematic measurement overheads. The effect of the parser modifications and instrumentation is to increase the execution time by about 50%; to compensate, we increase the copying overhead (by more than 50%) to 2 milliseconds to maintain the proper (conservative) ratio. All timings are derived from Quintus Prolog 1.5 running on a Sun 3/160 with 16 megabytes of memory.

Figure 6 shows a family of speed-up curves for the example sentence. The speed-up factor relative to the sequential execution time is shown as a function of the number of processors available. The additional parameter, which differs among the several curves, is the copying overhead, discussed in the next paragraphs. The straight line identifies 50% processor effectiveness, the points at which the speed-up factor is half the number of processors[6]. This arbitrary limit is taken as the threshold for acceptable system performance; it typically is at or near the knee of the speed-up curve. Table 4 summarizes the speed-up for the sentences parse. The "50% effectiveness" column reflects the

---

6. Processor utilization, which merely measures how busy the processors are, makes an apparent virtue of overhead. Processor effectiveness, sometimes called ***efficiency***, is the average ***useful*** processor utilization.

performance with the maximum number of processors which yields a processor effectiveness at or above 50%. (This number was not computed for fragment sentences, as the VCR did not properly handle the **xor** mechanism when these data were obtained.)

6.3. Effect of Overhead

Figure 6 shows the speed-up curves for our example sentence under five different copying overheads, ranging from 0 to 30 milliseconds; the process start-up is constant at 1 millisecond. As can be seen, if the copying time exceeds 4 milliseconds, more than a third of the potential speed-up is lost; at 10 milliseconds, more than half is gone. This suggests that parallelism can be effectively exploited if process overhead is kept to a fraction of the median process duration (here, 9 milliseconds).

Shen and Warren report simulation results [Shen1987] that show the same characteristic curves (their Figure 3) using an artificial time unit of a "resolution time", with the cost of process spawning usually being four resolutions (some tests vary it from 0 to 16 resolutions). They also find that the overhead has a substantial effect on the achievable speed-up. This leads us to hypothesize that the key parameter relating overhead to speed-up is the ratio of process initialization overhead to median process duration; this remains to be confirmed by additional experimentation.

PUNDIT Concurrency Results (2ms copying, 1ms startup)							
Sent no.	# wds	time(sec.) seq	par	maximum speedup	proc'rs	50% effectiveness speedup	proc'rs
5.1.2	4	10.8	0.9	11.8	62	-	-
1.1.1	5	7.3	0.7	10.1	42	9.9	20
25.1.3	6	26.8	1.5	18.5	121	-	-
31.1.4	6	24.9	1.1	23.2	71	-	-
31.1.3	7	6.1	0.9	7.1	28	6.9	14
6.1.2	7	7.8	0.9	8.5	37	7.9	16
22.1.1	7	22.5	1.9	11.6	59	-	-
4.1.3	8	7.4	1.2	6.4	37	5.8	12
9.1.1	8	13.7	1.2	11.9	42	11.6	23
9.1.4	11	37.3	1.4	26.1	93	22.6	45
4.1.1	12	51.5	1.6	31.8	153	27.8	56
28.1.1	14	46.2	1.9	24.3	116	21.1	42
- detailed analysis not available							

Table 4: Summary of Speed-Up Results.

6.4. Effect of Concurrent Nondeterministic Search

Spawning separate processes for grammatical disjuncts has the effect of starting the exploration of the "correct" parse sooner than would happen sequentially (unless it always derives from the left-most disjunct, which is first sequentially). Spawning ***all*** disjuncts has the effect that *for* any potential parse, there is a chain of processes that attempts that parse with ***no*** search overhead. This chain is, in effect, a nondeterministic search procedure that simply "guesses right" at each disjunction. Given sufficient resources, this nondeterministic search will accomplish the ultimate goal of reducing the time to develop the correct parse.

6.5. Effect of Sentence Length

Figure 8 shows the relationship between sentence length and parallel parsing time with unlimited processors. For these thirteen sentences, the relationship appears to be linear, which is consistent with the processing model. Since each potential parse is performed as quickly as possible, the time to complete it is proportional to the number of nodes in the parse tree. For these sentences, the number of nodes is apparently a linear function of the number of words, requiring about 10 nodes per word.

6.6. Linguistically Guided Concurrency

Our original intent was to investigate the use of static linguistic knowledge to guide spawning, e.g., spawn only on certain "rich" nodes, that is, nodes having several possible non-null expansions. Our motivation was to restrict parallel search to those alternative paths likely to result in substantial amounts of work. However, we then realized that a node expanding to a null option affects parsing by ***not*** removing a word from the word stream, causing a different search sequence to be followed. Thus even nodes expanding to the empty node produced substantial amounts of search, and we found that the simpler always-spawn policy was sufficiently effective that we have deferred investigation of other policies. Two contradictory effects are relevant to this issue:

- Spawning less frequently increases process granularity relative to overhead. As other efficiencies reduce process duration, it may be necessary to spawn more selectively in order to keep the spawning overhead from dominating the process duration.
- Fully nondeterministic search relies on the spawning of all disjuncts. Spawning more selectively will compromise the near-linear-time parsing we have obtained. In addition, since most of the disjuncts fail, the effect of spawning them is to remove processing from the successful parse paths; one may argue that this is desirable even if it increases processing overall.

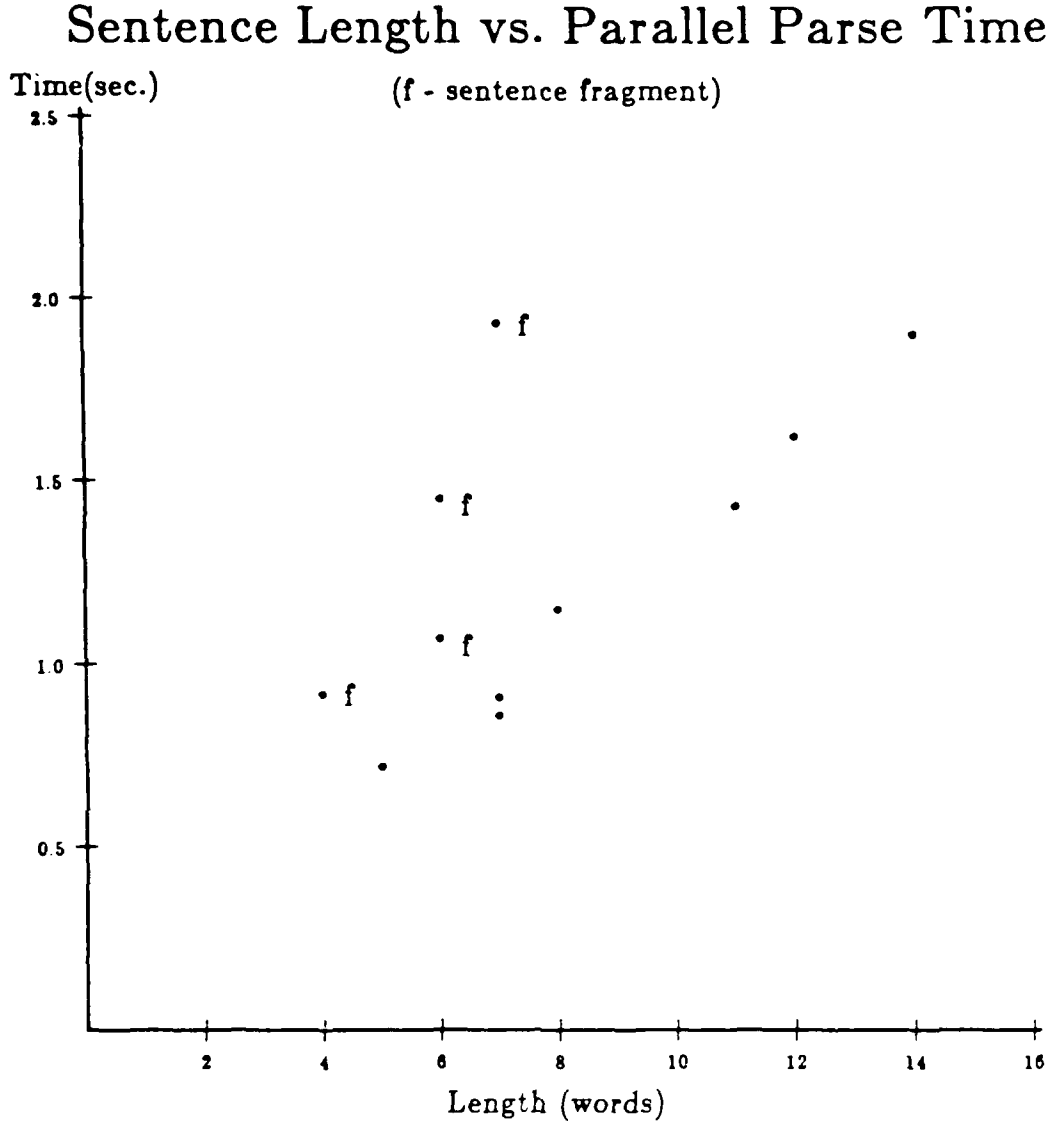


Figure 8: Parallel Parse Time

7. EFFECT OF IMPROVED SEARCH

The experiments described thus far used a somewhat simplified version of the PUNDIT parser. In this section we describe the effects of enabling the ***selection*** component of the system, which applies shallow semantic filtering to prune semantically anomalous constructions during parsing, and the ***xor*** mechanism, a sequential "or" connective used to control search when parsing incomplete or fragmentary sentences.

7.1. Selection

The selection component of PUNDIT performs tests on partial parse trees (corresponding to noun phrases and subject-verb-object relations) during the parsing process, to determine whether the structure is semantically plausible. When an assertion node is completed, selection examines the subject, verb, and object subtrees, extracts a subject-verb-object pattern and tests it for acceptability against a database of known patterns. For example, selection will accept the subject-verb-object combination ***engineer repairs compressor***, but reject the combination ***compressor repairs engineer*** in the domain of maintenance reports. Similarly within a noun-phrase, the system will examine the adjective-noun combination, accepting e.g., ***excessive wear*** but not ***excessive assembly***. The system maintains a database of known good patterns (sorted on syntactic type) and also of known bad patterns. These patterns are acquired via interaction with a domain expert and stored. The pattern database is used as a filter for parse trees, eliminating those that generate semantically invalid parses and accepting those that generate semantically valid ones.

Selection is applied to a partially built parse tree during parsing. It is applied at two places: upon completion of a noun-phrase, and upon completion of a subject-verb-object unit (e.g., assertion). If a noun-phrase is rejected, this can reduce the search space considerably; when a subject-verb-object pattern is rejected, this is often the last step in constructing it, so the number of parses is reduced, but the search space for single-clause sentences is not significantly reduced.

We investigated two interactions between selection and parallelism: the amount of parallelism available within selection processing, and the effect of sequential selection on parallelism under the existing model. The selection mechanism is deterministic, except for the very low-level parallelism in the final database access for matching good and bad patterns. Not surprisingly, we found little useful or-parallelism in the selection mechanism itself [7].

The effect of incorporating sequential selection in concurrent search is much more interesting. Much of the parallelism found in our previous work resulted from the high branching factor (bushiness) of the search space, and the relatively long computations involved in many of the attempted derivations. Noun-phrase selection is designed to prune away incorrect parses early in the parsing process, and would therefore be expected to reduce the achievable parallelism (which it does). It also

---

7. Interestingly, selection may be a good candidate for and-parallelism, because different constituents can be processed in parallel: e.g., the subject, the verb, and the object can all be examined in parallel, to generate the subject-verb-object pattern. This would require a different model, of course, and we make no claim about the utility of exploring this source of parallelism.

introduces new processing, and would be expected to increase the non-deterministic search time (but see below!).

The experiments required only simple changes to the parsing models, adding atoms to represent selection processing and generating new behavior descriptions from sequential runs with selection enabled. The rest of the methodology remained unchanged from the previous work.

In some limited testing, using sentences where selection is relatively effective, the results are quite dramatic. Table 5 shows the comparison between the parsing of one particularly flagrant sentence with and without selection. The sentence is a sentence fragment: ***Retained oil sample and filter element for future analysis.*** It can be parsed either correctly as a verb phrase (***someone retained sample and filter element***) or incorrectly as a noun phrase ***the retained sample and filter element***), both with variations on prepositional phrase attachment and scope of conjunction: ***oil (sample and filter) element*** vs. ***(oil sample) and (filter element)***, etc.

Selection, as expected, reduces the number of parses, in this case quite dramatically, from 22 parses to 1 parse. Not surprisingly, the amount of processing drops, as selection causes many search paths to be pruned away, including some that formerly succeeded. Also not surprising is the reduction in the amount of parallelism, since the pruned paths no longer run in parallel with others. What is surprising is the noticeable reduction in ***the time to termination***. Selectional filtering ***adds*** processing to the computation of each derivation, so we would expect the time to increase. Thus the observed decrease in time to completion took some further analysis to reconcile.

There is one important distinction that makes the discrepancy more comprehensible: the time to obtain all parses is not identical to the time to termination (exhaustion of all possibilities in the search space). In a parallel parse, several lines of search may continue for some time after the last valid parse is found; this effect can be seen in

	Without Selection	With Selection	Ratio
Parses Found	22	1	22
Sequential Time	127	46.0	2.76
Concurrent Time	2.66	2.43	1.09
Speed-up	48.0	18.9	2.54
Max Concurrency	265	110	2.41

Table 5: Effect of Selection on Parsing Behavior

Figure 9, where the dot on the abscissa indicates the time when a parse is completed. In the data shown above, selection has pruned off one or more paths that eventually failed but took longer than the successful parse paths. It therefore shortened the time to termination, despite increasing (slightly) the time for each successful parse.

There are several important conclusions to be drawn from this experiment:

- ***Better programs go faster*** in general. By making the parser more discerning in what it accepts as a valid parse, selection improves the quality of the program. In so doing, it makes the program run faster, both sequentially (dramatically) and in parallel (to a limited but significant degree). It also decreases the resource requirements for the parallel program. Given a fixed number of processors, the time to completion will be significantly shorter with selection. We take this as further evidence of a general rule that care in the design and implementation of a system will usually pay off.
- ***Improved search focus and concurrency are not incompatible.*** It may not be the case that improving the search focus reduces the speed up obtained by concurrency. The use of selection to prune the search space is a case in point. It reduces concurrency, but also reduces the time to terminate the search. Thus, while the speed-up factor is reduced with selection, as shown in Table 5, the speed-up relative to the sequential time without selection increases.

7.2. Treatment of Sequential-or (xor)

The domain of Navy messages contains many fragmentary sentences, that is, sentences missing a subject, as in ***Suspect faulty high speed rotating assembly***, or missing subject and verb, as in ***Loss of lube oil pressure during operation.*** We handle this [Linebarger1988] simply by adding the option ***fragment*** to the top level set of alternatives (***assertion***, ***question***, ***imperative***). We then specify the various fragment types in the definition for ***fragment***. However, this leads to an explosion in the search space, since there are five different fragment options, most of which begin with a noun-phrase. To avoid this explosion, PUNDIT has a special sequential "or" connective, ***xor***, that allows us to try for the "most structured" alternative first. Only if no complete sentence (e.g., assertion, question or imperative) is found do we try the fragment option. Once in the fragment option, however, we need to find ***all*** fragments. The ***xor*** connective supports this with following semantics (although it is not implemented in this way, for efficiency reasons):

A xor B :- A $\longrightarrow$ A; B.

Use of ***xor*** has the effect of introducing a sequential operation into the parsing: we wait until ***A*** terminates (producing no solutions) before

beginning B^8.

The sequential-or construction complicates the models somewhat, since the sequential semantics must be maintained in the concurrent execution model. The ordering information that must be captured in the model reflects the requirement that ***all*** possible parses in the first option must fail **before** **xor** will examine the second. A simple extension of the model to support success and failure completion signal propagation is sufficient to model **xor**. Each process, instead of simply terminating on success or failure, reports success in a completion signal to its parent process if it or any of the processes it spawned has successfully parsed the sentence; otherwise, it reports failure. The completion signals are defined recursively down the process tree, with a terminal node reporting success if and only if it matches a word in its category.

To implement completion signals, each process has an epilogue in which it waits for all of its children to signal completion, and signals completion to its parent only when it and all its children are complete. Each process, after completing its searching, thus remains in the system until each of its children has finished its epilogue. **xor** can thus examine the completion signal of the first option to decide whether to invoke the second.

Our simulations for fragment sentences model this behavior. Parses for grammatical sentences complete in the first **xor** alternative, and completion signal propagation affects no behavior except process termination. The duration of the epilogue processing is sufficiently small to be negligible, as is the implication of the extended process lifetime, discussed in Section 8.1.

Figure 9 is the concurrency graph for the parse of the fragmentary sentence discussed under Selection, Section 7.1. The effect of **xor**'s waiting for complete failure of the more structured options shows in the long period of near idleness between 900 and 1200 milliseconds. Then, when failure finally propagates up to **xor**, the ***fragment*** option is invoked, and a great deal of concurrency results almost immediately. Since this run was with selection, only the correct solution is found, at about 2200 milliseconds.

---

8. While **xor** could evaluate both *A* and *B* in parallel, its semantics would nonetheless require that any solution from *B* be held until *A* had found a solution or had terminated. This speculative kind of approach might be useful with a more sophisticated scheduling strategy, when a large number of processors might otherwise be idle.

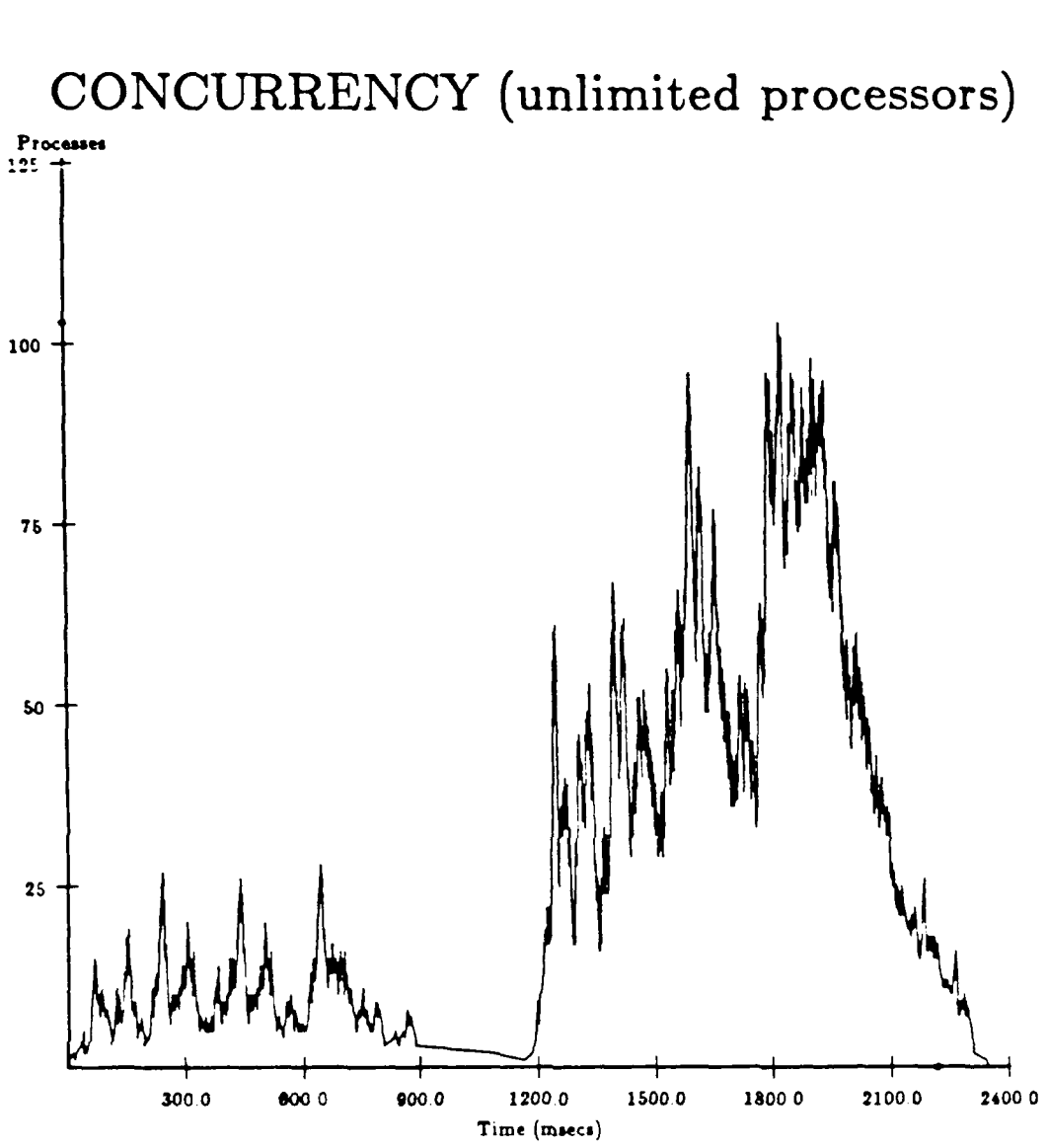


Figure 9: Fragment Parse with Selection

8. ARCHITECTURAL IMPLICATIONS

Thus far we have described a model of parallel execution for the PUNDIT parser, asserted run-time costs for the execution model, and shown the performance over a range of such costs. In this section we examine the architectural implications of the model, sketch some key aspects of an implementation, and compare it with more general models of or-parallel logic programming, particularly of Prolog. We take the Argonne model of or-parallel Prolog [Shen1987] as being representative of a number of such models.

It is important to realize that this is an apples-to-oranges comparison. Our model, developed to fit one application, supports or-parallelism only with strong restrictions on the occurrence of shared unbound variables. This vastly simplifies the handling of variables, which is one of the key parts of most models of Prolog or-parallelism. By tailoring the execution model to the application, we are able to show higher speed-ups than we would expect with more general models.

8.1. Processes, Processors, and Memory

A major difference between our *model* and most or-parallel Prolog models is the way in which processes are defined, and how they are assigned to processors. Because we identify the work to be done by each process statically (at compilation time), there is no run-time overhead to do this. The associated cost is the need to handle a variable and occasionally large number of processes in the system. The Argonne model, in contrast, limits the number of processes to the number of processors, avoiding dynamic process scheduling, but must determine dynamically how to divide the work among the processes. In general, static process identification is difficult, especially in search-dominated AI applications, where the amount of processing associated with a search path is data-dependent and essentially unknowable ***a priori***. The specifics of the PUNDIT parser's behavior, however, allow us to know that the granularity, in particular, of statically defined processes will be acceptable. We thus avoid the overheads of estimating the amount of "work" in identifiable parts of the computation and dynamically partitioning the work.

Scheduling of processes, while more difficult than the static assignment of Argonne-like models, is relatively easy, since we have not yet seen the need to go beyond a very simple FIFO scheduler to priority-based scheduling. This may well change with further insight into the behavior of systems with better search heuristics or tighter constraints on resources and response time, but it does not appear that the associated computation will be onerous. We certainly do not expect the run-time cost of process scheduling to approach the run-time cost of dynamic process identification.

Memory utilization is another important measure of a system. Without pre-emptive scheduling (which would allow a running process to lose its processor to a higher priority process), our model requires only a very simple memory usage analysis. Since a process, once running, doesn't relinquish its processor, the number of processes which are in the middle of an active computation is limited to the number of processors. We can characterize the memory usage of running and queued processes:

- Each queued process will use only enough memory to describe its initial state (as described in section 4.3). This amounts typically to a

few hundred bytes. When it gains a processor, it will initialize a larger area to support the Prolog stack and other data areas. For the sentences simulated, the number of queued processes peaks at no more than a few hundred, requiring a small fraction of a megabyte of storage.

- Each running process will use a predictable amount of memory for stack and allocated data items which is essentially the same amount used for this data by conventional sequential systems.
- Code, of course, is re-entrant and shared; a single copy serves all processes.

Adding termination processing, as discussed in 7.2, increases the needs somewhat, as more processes may be in "running" format. A process that has finished its assigned work and is waiting for termination signals from the processes it spawned, however, can shrink back to a small memory area; it requires only the termination signal areas and enough information to integrate it and pass the appropriate signal to its spawner. This memory requirement will approximate the requirement for queued processes; the two together will still be well under a megabyte. The number of processes requiring large amounts of memory, then, will remain bounded by the number of processors. This conclusion doesn't hold, of course, for a priority-based pre-emptive scheduler, which may have arbitrarily many processes in the running state, most of them pre-empted, and therefore does not have such benign memory usage behavior. More sophisticated control of the search process is needed in this case, and is a current topic of research.

8.2. Hardware Requirements

Throughout this research, we have used real multiprocessor systems as a guide in designing the system, which we believe to be readily implementable on a number of machines. We have taken the Encore Multimax [Encore1986] and the Mach operating system [Accetta1986] as our principal target; other shared memory multiprocessors, including systems from Sequent, Alliant, etc., can also effectively support the concurrent execution model. This section addresses the specific architectural requirements that contributed to that choice. Specific performance estimates for our simulations that cannot be drawn from our Sun workstation measurements have used the Multimax/Mach system performance.

The amount of code in PUNDIT is large, the locality of reference is typically low, and the allocation of processes to processors is unpredictable, making it difficult to partition the code for memories local to each processor. A ***global shared memory*** avoids this problem and the limitation of local memory size. Shared data accesses, while less frequent than code references, are made by all processes; a shared memory

architecture supports this requirement well.

Code is fixed, and shared data changes infrequently, so there will not be much contention for exclusive access to shared memory locations. A ***coherent local cache system*** will have few data invalidations, and can achieve a very high hit rate; this lowers global memory and memory bus usage. Systems without a coherent cache system must have sufficient memory and bus bandwidth to support the processors used; this may limit the number of processors for such systems.

The choice of a ***processor*** is based essentially on its conformity to the system hardware architecture requirements, which we do not address, and its ability to support Prolog well. The trend in compilation technology for Prolog is to compile away much of the specialized "Prolog-ish" structure, with the possible exception of choicepoints. Since parallelism obviates the need for these, we do not believe that specialized processors will have any lasting advantage over general-purpose processors. It will be difficult for a specialized processor to remain cost-effective because of the difficulty in keeping its VLSI implementation competitive with commercial microprocessor offerings.

8.3. Operating System Requirements

The support of the parallel execution model poses some significant challenges to the operating system in the areas of process creation and scheduling, as well as memory sharing. Mach supports the concept of ***threads***, or "light-weight" tasks, operating within a single addressing environment, which appear to match well with our process model. Specific requirements include the following:

- The system must be able to create and initialize a process quickly. The Multimax/Mach time to create a thread is less than 200 microseconds [Chen1987], which is smaller than the variation in our measurements of data initialization time.
- The system must be able to handle hundreds of queued processes and dispatch them for execution efficiently. Mach appears to have this capability, as there is no need to change contexts when switching among the threads, and the number of threads is not inherently limited. The process queue is potentially a system bottleneck with large numbers of processors; it may be necessary to use a hierarchical queue system, with groups of processors usually sharing "local" queues, for systems with Order(100) processors
- Memory management must be able to supply shared and private data segments. The parallel execution model assumes that each process's Prolog stack will be private, and that the parse tree and continuation stack will be sharable with other processes. Since the sharable data items can be allocated piecemeal, it is not necessary to have a separate sharable area for each process (although this has

a significant effect on storage reclamation, as discussed below).

Storage allocation of sharable memory in a parallel system differs from sequential Prolog storage allocation because the parallelism violates the pure stack discipline that allows the normal Prolog "heap" allocation pointer to be reset when returning to a choicepoint. Since an item in memory may be shared among many processes, it cannot be reclaimed until all such processes have been terminated. In addition, if the item becomes part of a successful parse tree, it is then accessible from the global solution list, and must be preserved for succeeding stages of linguistic processing. It is thus not useful to partition the heap into separate areas for each process, since items in any such area may need to persist well past the end of the process lifetime.

8.4. Specialized Support Software

The final layer of software supports the Prolog and parallelism operations of the parallel PUNDIT system. In general, this is a standard sequential Prolog system modified for parallel execution using the parallelism primitives of Mach. Areas requiring substantial changes are the elimination of choicepoint recording for disjuncts evaluated in parallel, and the implementation of a storage manager that supports shared access to dynamic data. We do not consider the first area to have any fundamental problems, as it is a straightforward matter to use the choicepoint mechanism to drive process spawning. The rest of this section discusses dynamic storage allocation and garbage collection.

The process model provides a simple dichotomy of storage as strictly private to a process or sharable among arbitrarily many processes. Sharable items are the word list, the overlapping forest of parse trees. and the overlapping (cactus) continuation stacks. They present no great problem in allocation as they can easily be mapped into a (conceptually) flat shared address space (however it is implicitly structured in an implementation). The Prolog implementation must be extended to allow allocation of both private and sharable dynamic data. The private area is used for the Prolog stack and the Prolog "heap" of private dynamic data. It too presents no problem in allocation; there are no pointers into it from outside, and it can be deallocated when the process terminates. Process information, kept in a separate area by the operating system, includes the queued/active/terminating state designator and other process pointers, all of which are statically allocated and cannot point into data areas.

The remaining question is storage reclamation, or garbage collection, in sharable memory. We have assumed that the trees for correct parses are maintained on a single "solution list" in sharable memory, and that partial trees for parses in progress are accessible through the processes constructing them. Any of the various mark and sweep techniques can be used to reclaim unreachable data. Sharable data must be

preserved if it can be reached from any live process (active or queued), or from the solution list. Once a process has either succeeded (posting its tree on the solution list) or failed, it has no further need to access sharable memory, or to force its preservation; such "terminating" processes relinquish their private data areas and need not be considered any further by the sharable storage manager.

A mark/sweep garbage collector can be implemented along any of several conventional lines. The use of a so-called "real time" garbage collector, one that runs in parallel with the parser, will prevent the hiatus in processing associated with the more traditional variety, at some cost in complexity. In either case, the marking phase must trace each running or queued process to find its pointers into sharable memory, and any structures referenced by them. In addition, of course, the solution list must also be traced.

The use of reference counts for identifying reclaimable storage also appears possible. Only a few structures are built dynamically in sharable storage, as detailed in section 5.3, and there are presently no circular structures built, so reference counting techniques appear attractive. Their use, however, depends much more than the rest of the system on the details of the parsing algorithms and their implementation; the implications of the use of reference counts require much more careful study.

9. CONCLUSIONS

We have demonstrated exploitable or-parallelism in an important real Prolog application, and have simulated significant speed-up with an application-specific parallel execution model. For a small but representative set of sentences, we have observed linear-time parsing in the unlimited-processor case. We have also shown an interesting relationship between search focus and concurrency, indicating that good sequential search heuristics benefit concurrent execution.

Some necessary work remains to sharpen and improve the simulation model and parameters. We have previously mentioned the instrumentation overhead that remains in the atom duration measurements. We have also not treated garbage collection properly, as we have not yet dug into the Prolog implementation to identify and measure it precisely to allow prediction of a multiprocessor equivalent. The overhead of maintaining choice points and trail information, unnecessary for spawned disjuncts, is another unknown. These effects may reduce the granularity of the processes, making the overhead more significant if it is not further optimized. Finally, the measurements are made on a parser that interprets the grammar, rather than running the more efficient version of the grammar obtained by translating it to Prolog.

This typically doubles the parser speed, so speed-up projections for this case will be consistent with the results reported here for the 4 millisecond copying cost.

The results described here apply to an application-specific large-grained form of concurrency. Research in related areas [Blenko1988, Shen1987] points to the existence of finer-grain concurrency within the general or-parallel model that would apply to the computations that we have taken as atomic. Further research is necessary to determine the amount of additional speed-up that can be gained by combining approaches.

We have recently begun collaboration with the researchers at the Swedish Institute of Computer Science; we have furnished the application described here for benchmarking on the Aurora Or-parallel machine [Hausman1987]. Preliminary results have been extremely encouraging and lead us to believe that our estimates of useful or-parallelism are quite conservative. Further experiments and detailed analysis are now underway. This work should provide us with the ability to validate our simulation results, as well as providing insight into speed-ups obtainable on large-scale applications in a general or-parallel implementation.

REFERENCES

[Accetta1986]
Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young, Mach: a new kernel foundation for UNIX development. In ***Proceedings of Summer Usenix***, CMU Computer Science Department Technical Report, July, 1986.

[Bates1983]
P. Bates and J. Wileden, High-level debugging of distributed systems: The behavioral abstraction approach, Technical Report COINS 83-29, Dept. of Computer and Information Science, Univ. Massachusetts, 1983.

[Blenko1988]
T. Blenko, Compiling Logic Programs to Applicative Form, LBS Technical Memo 74, Paoli Research Center, Unisys Corporation, Paoli, Pennsylvania, January, 1988.

[Chen1987]
Pin-Yee Chen, , personal communication, August, 1987.

[Ciepielewski1985]
A. Ciepielewski, S. Haridi, and B. Hausman, Initial Evaluation of a Virtual Machine for OR-Parallel Execution of Logic Programs. ***Proceedings of IFIP TC-10***, U. of Manchester , 1985.

[Conery1987]
J. S. Conery, Binding Environments for Parallel Logic Programs in Non-Shared Memory Multiprocessors. In ***Proc. of the 1987 Symposium on Logic Programming***, San Francisco, 1987, pp. 457-467.

[Disz1987]
Terry Disz, Ewing Lusk, and Ross Overbeek, Experiments with Or-Parallel Logic Programs. In ***Proc. of the Third International Conference on Logic Programming***, J. L. Lassez (ed.), Melbourne, 1987, pp. 576-600.

[Dowding1987]
John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar. In ***Proc. of the Second International Conference on Natural Language Understanding and Logic Programming***, Vancouver, August, 1987.

[Encore1986]
Encore, ***Multimax Technical Summary***. Encore Computer Corporation, Marlboro, Massachusetts, September 1986.

[Fagin1987]
B. S. Fagin and A. M. Despain, Performance Studies of a Parallel PROLOG Architecture. In ***Proceedings of the 14th International Symposium on Computer Architecture***, Pittsburgh, Pennsylvania, June, 1987, pp. 108-116.

[Hausman1987]
B. Hausman, A. Ciepielewski, and S. Haridi, OR-Parallel Prolog Made Efficient on Shared Memory Multiprocessors. In ***Proc. of the 1987 Symposium on Logic Programming***, San Francisco, 1987, pp. 69-79.

[Hirschman1989]
Lynette Hirschman, Martha Palmer, John Dowding, Deborah Dahl, Marcia Linebarger, Rebecca Passonneau, Francois Lang, Catherine Ball, and Carl Weir, The PUNDIT Natural Language Processing System. In ***Proc. of the Conference on Artificial Intelligence Systems in Government***, Washington, D.C., March, 1989, pp. 234-243.

[Hirschman1985]
L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In ***Logic Programming and its Applications***, D.H.D. Warren and M. VanCaneghem (ed.), 1985, pp. 244-261.

[Hirschman1986]
L. Hirschman, Conjunction in Meta-Restriction Grammar. ***J. of Logic Programming*** **4**, 1986, pp. 299-328.

[Hirschman1988]
L. Hirschman, A Meta-Treatment of Wh-Constructions in English. In ***Proc. of META88, Meta-Programming in Logic Programming***, Bristol, UK, June, 1988.

[Hopkins1987]
W. Hopkins and R. Smith, Concurrency Simulation by Abstract Interpretation, LBS Technical Memo, Paoli Research Center, Unisys Corp., Paoli, Pennsylvania, December, 1987.

[Lang1988]
F.-M. Lang and L. Hirschman, Improved Parsing Through Interactive Acquisition of Selectional Patterns. In ***Proc. of the Second Conference on Applied Computational Linguistics***, Austin, Texas, February, 1988.

[Linebarger1988]
M. Linebarger, D. Dahl, L. Hirschman, and R. Passonneau, Sentence Fragments Regular Structures. In ***Proc. of the 1988 Annual Conference on Computational Linguistics***, Buffalo, New York, June, 1988, pp. 7-16.

[Moto-oka1984]
T. Moto-oka, H. Tanaka, H. Aida, and T. Maruyama, The Architecture of a Parallel Inference Engine - PIE. In ***Proceedings of the International Conference on Fifth Generation Computer Systems***, 1984, pp. 479-488.

[Pereira1980]
F. C. N. Pereira and D. H. D. Warren, Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks. ***Artificial Intelligence*** **13**, 1980, pp. 231-278.

[Shapiro1987]
Ehud Shapiro, An Or-Parallel Execution Algorithm for Prolog and its FCP Implementation. In ***Proc. of the Third International Conference on Logic Programming***, J.-L. Lassez (ed.), Melbourne, Australia, 1987, pp. 311-337.

[Shen1987]
K. Shen and D. H. D. Warren, A Simulation Study of the Argonne Model for Or-Parallel Execution of Prolog. In ***Proc. of the Third International Conference on Logic Programming***, J.-L. Lassez (ed.), Melbourne, Australia, 1987, pp. 54-68.

[Sohma1985]
Y. Sohma, K. Satoh, K. Kumon, H. Masuzawa, and A. Itashiki, A New Parallel Inference Mechanism Based on Sequential Processing, TM-0131, ICOT, 1985.

[Warren1987]
D. H. D. Warren, The SRI Model for Or-Parallel Execution of Prolog -- Abstract Design and Implementation. In ***Proc. of the 1987 Symposium on Logic Programming***, The Computer Society of the IEEE, San Francisco, 1987, pp. 92-102.

[Westphal1987]
H. Westphal and P. Robert, The PEPSys Model: Combining Backtracking, AND- and OR-Parallelism. In ***Proc. of the 1987 Symposium on Logic Programming***, San Francisco, 1987, pp. 436-448.

Lang, F., and Hirschman, L."Improved Portability and Parsing Through Interactive Acquisition of Semantic Information". *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas, February, 1988.

IMPROVED PORTABILITY AND PARSING THROUGH INTERACTIVE ACQUISITION OF SEMANTIC INFORMATION*

François-Michel Lang and Lynette Hirschman

Paoli Research Center, UNISYS
P. O. Box 517, Paoli, PA 19301

ABSTRACT

This paper presents SPQR (Selectional Pattern Queries and Responses), a module of the PUNDIT text-processing system designed to facilitate the acquisition of domain-specific semantic information, and to improve the accuracy and efficiency of the parser. SPQR operates by interactively and incrementally collecting information about the semantic acceptability of certain lexical co-occurrence patterns (e.g., subject-verb-object) found in partially constructed parses. The module has proved to be a valuable tool for porting PUNDIT to new domains and acquiring essential semantic information about the domains. Preliminary results also indicate that SPQR causes a threefold reduction in the number of parses found, and about a 40% reduction in total parsing time.

1. INTRODUCTION

A major concern in designing a natural-language system is portability: It is advantageous to design a system in such a way that it can be ported to new domains with a minimum of effort. The level of effort required for such a port is considerably simplified if the system features a high degree of modularity. For example, if the domain-independent and domain-specific components of a system are clearly factored, only the domain-specific knowledge bases need be changed when porting to a new domain. Even if a system demonstrates such separation, however, the problem remains of ***acquiring*** this domain-specific knowledge.

One obvious benefit of acquiring domain-specific semantic information is rejecting parses generated by the syntactic component which are semantically anomalous. Using domain knowledge to rule out semantically anomalous parses is especially important when parsing with large, broad-coverage grammars such as ours: Our Prolog implementation of Restriction Grammar [Hirschman1982,Hirschman1985] includes about 100 grammar rules and 75 restrictions, and is based on Sager's Linguistic String Grammar [Sager1981]. It also includes a full treatment of sentential fragments and telegraphic message style. As a result of this extended coverage, many sentences receive numerous syntactic analyses. A majority of these analyses, however, are incorrect because they violate some semantic constraint.

Let us take as an example the sentence ***High lube oil temperature believed contributor to unit failure.*** Two of the parses for this sentence could be paraphrased as:

(1) The high lube oil temperature believed the contributor to the unit failure.

(2) The high lube oil temperature was believed to be a contributor to the unit failure.

but our knowledge of the domain (and common sense) tells us that the first parse is wrong, since temperatures cannot hold beliefs.

It is only because of this semantic information that we know that parse (2) is correct, and that parse (1) is not, since we cannot rule out parse (1) on syntactic grounds alone. In fact, our grammar generates the incorrect parse before the correct one, since it produces full assertion parses before fragment parses. If the syntactic component has access to semantic knowledge,

*This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research (APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED), in part by National Science Foundation contract DCR-85-02205, as well as by Independent R&D funding from System Development Corporation, now part of Unisys

however, many incorrect parses such as (1) will never be generated.

How then can we collect the necessary semantic information about a domain? One traditional approach involves analyzing a corpus of texts by hand, or perhaps even simply relying on one's intuitive knowledge of the domain in order to gather information about what relations can hold among domain entities. Several obvious drawbacks to these approaches are that they are time-consuming, error-prone, and incomplete. A more robust approach would be to use (semi-) automated tools designed to collect such information by cataloguing selectional patterns found in correct parses of sentences.

However, our reasoning appears circular: The desired domain-specific information can only be obtained from analyses of correctly parsed sentences, but our goal is to restrict the parser to these correct analyses precisely by using this domain knowledge. In the example above, we need the semantic knowledge to rule out the first parse; but it is only by knowing that this parse is semantically anomalous that we can obtain the selectional information about the domain.

One way to avoid this circularity is to bootstrap into a state of increasingly complete domain knowledge. We have implemented in SPQR such a bootstrapping process by incrementally collecting and storing domain-specific data gathered through interaction with the user. The data are in the form of selectional constraints expressed as allowable and unallowable syntactic co-occurrence patterns. All the data collected while parsing a set of sentences can then be used to help guide the parser to correct analyses and to decrease the search space traversed during future parsing. As the system's semantic knowledge becomes increasingly rich, we can expect it to demonstrate some measure of learning, since it will produce fewer incorrect analyses and present fewer queries to the user about the validity of syntactic patterns.

A number of systems have been developed to assist the user in acquiring domain-specific knowledge including TELI [Ballard1986], TEAM [Grosz1983], KLAUS [Hendrix1980], ASK [Thompson1983] and [Thompson1985], TQA [Damerau1985] and IRACQ [Moser1984, Ayuso1987]. Related work has also been reported in [Tomita1984], as well as in [Grishman1986] and [Hirschman1986a]. The work described here differs from these previous efforts in several ways:[1]

- Since PUNDIT is not a natural-language interface or a database front end, but rather a full text-processing system, sentences analyzed by PUNDIT are taken from corpuses of naturally-occurring texts. The semantic information gathered is therefore empirically or statistically based, and not derived from sentences generated by a user.
- The elicitation of information from the user follows a highly structured, data-driven approach, yielding results which should be more reproducible and consistent among users.
- Many systems have a clearly defined knowledge-acquisition phase which must be completed before the system can be effectively used or tested. We have chosen instead to adopt a paradigm of incremental knowledge acquisition.

Our incremental approach is based on the assumption that gathering complete knowledge about domain is an unattainable ideal, especially for a system which performs in-depth analysis of texts written in technical sublanguages: Even if one could somehow be assured of acquiring all conceivable knowledge about a domain, the system's omniscience would be transient, since the technical fields themselves are constantly changing, and thus require modifications to one's knowledge base. An incremental acquisition method therefore allows us to start from an essentially empty knowledge base. Each sentence parsed can add information about the domain, and the system thereby effectively bootstraps itself until its knowledge about the selectional patterns in a domain approaches completeness.

In this paper we present SPQR, the component of the PUNDIT[2] text-understanding system which is designed to acquire domain-specific selectional information [Lang1987]. We present in Section 2 the methodology we have adopted to collect and use selectional patterns, and then give in Section 3 some examples of the operation of our

[1] See [Ballard1986] for a detailed and informative comparison of TELI, TEAM, IRACQ, TQA, and ASK.

[2] PUNDIT (Prolog UNDerstands Integrated Text) is implemented in Quintus Prolog, and has been described in [Hirschman1985] and [Hirschman1986b] (syntax), [Palmer1986] (semantics), [Dahl1986] (discourse), and [Passonneau1986] (temporal analysis).

module. We conclude by presenting some experimental results and discussing some future plans to extend the module.

SPQR has been used in analyzing texts in three domains: casualty reports (CASREPs) dealing with mechanical failures of ***starting air compressors*** (SACs are a component of a ship's engine), queries to a Navy ships database, and Navy sighting messages (RAINFORMs).

2. METHODOLOGY

The essential feature of our parser which facilitates the collecting of syntactic patterns is the INTERMEDIATE SYNTACTIC REPRESENTATION (ISR) produced by the syntactic analyzer. The ISR is the result of regularizing the surface syntactic structure into a canonical form of operators and arguments. Since there are only a limited number of structures which can appear in an ISR, we have been able to write a program to analyze the ISR and examine the syntactic patterns as they are generated.

```
[past,repair,
   [tpos(the),
      [nvar([engineer,singular,_])]],
   [tpos(the),
      [nvar([sac,singular,_])],
      adj([pastpart,break])]]
```

Figure 1: ISR for the sentence
The engineer repaired the broken sac

A brief note about the implementation: Since the ISR is represented as a Prolog list, the program which analyzes it was written as a definite-clause grammar and has the flavor of a small parser. As a sample ISR, we present in Figure 1 the regularized representation of the obvious parse for the sentence ***The engineer repaired the broken sac*** (pretty-printed for clarity). At the top level, the ISR consists of the main verb (preceded by its tense operators), followed by its subject and object. The ISR of a noun phrase contains first the determiner (labelled TPOS), then the head noun, (the label NVAR stands for "noun or variant"), and finally any nominal modifiers. Note that part of the regularization performed by the ISR is morphological, since the actual lexical items appearing in the ISR are represented by their root forms. Hence ***broken*** in the input sentence is regularized to ***break*** in the ISR, and ***repaired*** in the input sentence appears in the ISR simply as ***repair***.

SPQR is invoked by two restrictions which are called after the BNF grammar has assembled a complete NP (and constructed the ISR for that NP), and after it has assembled a complete sentence (and constructed its ISR). The program operates by presenting to the user a syntactic pattern (either a head-modifier pattern or a predicate-argument pattern) found in the ISR, and querying him/her about the acceptability of that pattern.

For each of the basic types of patterns which the program currently generates, the chart in Table 1 shows that pattern's components, an example of that pattern, and a sentence in which the pattern occurs. When presented with a syntactic pattern such as those in the chart in Table 1, the user can respond to the query in one of two ways, depending on the semantic compatibility of the predicate and arguments (e.g., in the case of an SVO pattern) or of the head and modifiers (e.g., for an ADJ pattern) contained in the pattern. If the pattern describes a relationship that can be said to hold among domain entities (i.e., if the pattern occurs in the sublanguage), the user accepts the pattern, thereby classifying it as good. The analysis of the ISR and the parsing of the sentence are then allowed to continue. If, however, the pattern describes a relationship among domain entities that is not consistent with the user's domain knowledge or with his/her pragmatic knowledge (i.e., if the pattern cannot or does not occur in the sublanguage) the user rejects it, thereby classifying it as bad, and signalling an incorrect parse. This response causes the restriction which checks selection to fail, and as a result, the parse under construction is immediately failed, and the parser backtracks.

As the user classifies these co-occurrence patterns into ***good patterns*** and ***bad patterns***, they are stored in a pattern database which is consulted before any query to the user is made. Thus, once a pattern has been classified as good or bad, the user is not asked to classify it again. If a pattern previously classified as bad by the user is encountered in the course of analyzing the ISR, SPQR consults the database, recognizes that the pattern is bad, and automatically fails the parse being assembled. Similarly, if a pattern previously

TABLE 1: Selectional Patterns

PATTERN	COMPONENTS	EXAMPLE
(1) SVO	subject, main verb, object	inspection reveal particle
INSPECTION of lube oil filter REVEALED metal PARTICLES.		
(2) ADJ	adjective, head*	normal pressure
Troubleshooting revealed NORMAL sac lube oil PRESSURE.		
(3) ADV	head, adverb	decrease rapidly
Sac air pressure DECREASED RAPIDLY to 5.74 psi.		
(4) CONJ	$conjunct_1$, conjunction, $conjunct_2$	pressure and temperature
Troubleshooting revealed normal PRESSURE AND TEMPERATURE.		
(5) NOUN-NOUN	noun modifier, head	valve part
VALVE PARTS excessively corroded.		
(6) PREP	head, prep, object	disengage after alarm
DISENGAGED immediately AFTER ALARM.		
(7) PREDN	noun, predicate nominal	capability necessity
Alarm CAPABILITY is a NECESSITY.		

*We use "head" throughout the chart to denote the head of a construction in which a modifier appears. The head can simply be thought of as that word which the modifier modifies.

recorded as good is encountered, SPQR will recognize that the pattern is good simply by consulting the database, and allow the parsing to proceed.

The selectional mechanism as described so far deals only with lexical patterns (i.e., patterns involving specific lexical items appearing in the lexicon). However, we have implemented a method of generalizing these patterns by using information taken from the domain ***isa*** (generalization/specialization) hierarchy to construct semantic class patterns from the lexical patterns. After deciding whether a given pattern is good or bad, the user is asked if the relation described by the pattern can be generalized. In presenting this second query, SPQR shows the user all the super-concepts of each word appearing in the pattern, and asks for the most general super-concept(s), if any, for which the relation holds.

Let us take as an example the noun-noun pattern generated by the compound nominal ***oil pressure***. While parsing a sentence containing this expression, the user would accept the noun-noun pattern [***oil, pressure***]. The program will then show the user in hierarchically ascending order all the generalizations for ***oil*** (***fluid***, ***physical_object***, and ***root_concept***), and all the generalizations for ***pressure*** (***scalar_quantity***, ***object_property***, ***abstract_object***, and again ***root_concept***). The user can then identify which of those super-concepts of ***oil*** and ***pressure*** can form a semantically acceptable compound nominal. In this case, the correct generalization would be [***fluid, scalar_quantity***], because

- The fluids in the domain are oil, air, and water; the scalar quantities are pressure and temperature; and it is consistent with the domain to speak of the pressure and the temperature of oil, air, and water.
- We cannot generalize higher than ***fluid*** since it would be semantically anomalous to speak of "physical_object pressure" for every ***physical object*** in the domain (e.g., one

would not speak of ***connecting_pin pressure*** or ***gearbox pressure***).

- We cannot generalize higher than ***pressure*** since ***shape*** is also an ***object_property***, and it would be infelicitous to speak of ***oil shape***.

As with the lexical-level patterns, the user's generalizations are stored for reference in evaluating patterns generated by other sentences. The obvious advantage of storing not just lexical patterns but also semantic patterns is the broader coverage of the latter: Knowing that the semantic class pattern [***fluid, pressure***] is semantically acceptable provides much more information than knowing only that the lexical pattern [***oil, pressure***] is good.

3. SOME (SIMPLIFIED) EXAMPLES

As we mentioned earlier, multiple syntactic analyses which can only be disambiguated by using semantic information abound in our corpuses because of the telegraphic and fragmentary nature of our texts. This ambiguity has two principal causes:

(1) A sentence which parses correctly as a fragment can often be parsed as a full assertion as well.

(2) Determiners are often omitted from our sentences, thus making it difficult to establish NP boundaries.

Since such syntactically degenerate sentences will generally contain fewer syntactic markers than full, non-telegraphic English sentences, they are characterized by correspondingly greater ambiguity. We now present an example of the use of selection to rule out a semantically anomalous assertion parse in favor of a correct fragment reading.[3] Consider the sentence ***Loss of second installed sac.*** In the correct analysis, the sentence is parsed is a noun string fragment; however, another reading is available in which the sentence is analyzed as a full assertion, with ***loss of second*** as the subject, ***installed*** as main verb, and ***sac*** as direct object. A paraphrase of this parse might be ***The loss of a second installed the sac.*** But this analysis is semantically completely anomalous for several reasons, but most notably because it makes no sense to say that the loss of a second can cause a sac (or anything else) to be installed. Since our parser tries assertion parses before fragment parses, the incorrect reading of this sentence is produced first. In generating the assertion parse, the parser encounters the SVO pattern [***loss, install, sac***], and queries the user as follows:

```
<SVO> pattern : loss install sac
```

This query asks if a loss can install a sac in this domain, or if a domain expert would ever speak of a loss installing a sac. Since it is nonsensical to speak of a loss installing a sac, the correct response to SPQR's query in this case is to reject the pattern, causing the assertion parse to fail after the module elicits the appropriate generalizations of the pattern.

In order to generalize the pattern, the user is shown all the the super-ordinates of ***loss*** and ***sac***, and asked to generalize the anomalous SVO pattern [***loss, install, sac***]. The super-concepts of ***loss*** are ***failure***, ***problem***, ***event***, ***abstract_object***, and ***root_concept***. The super-concepts of ***sac*** are ***unit***, ***mechanical_device***, ***system_component***, ***physical_object***, and ***root_concept***. Since nothing that is an abstract object can install anything at all, the correct generalization would be [***abstract_object, install, root_concept***]. Since the user's response to the original prompt labelled the pattern as bad, the assertion parse under construction then fails, and the parser backtracks.

An especially convoluted example of assertion-fragment ambiguity is found in the sentence ***Experienced frequent losses of pressure following clutch engage command.*** In the correct reading (which is again a fragment), the subject is elided, the main verb is ***experienced***, and the direct object is the ***frequent losses of pressure*** (in this parse, ***following clutch engage command*** functions as a sentence adjunct, with ***clutch engage command*** as a compound nominal). However, in another reading generated by our parser, the subject is ***experienced frequent losses of pressure following clutch***, the main verb is ***engage***, and ***command*** is the direct object. This reading would fail selection at the SVO level (if not sooner) because the SVO pattern [***loss, engage,***

[3] In this simplified explanation, we present only the SVO pattern. In actual parsing of this sentence, however, additional patterns would be generated from the NP level.

TABLE 2: Statistical Summary of 31 Sentences

PARSING INFORMATION	WITH SPQR	W/OUT SPQR
# of sentences receiving a correct parse	31	29
# of sentences receiving a correct FIRST parse	30	17
# of sentences receiving more than one parse	8	22
average # of parses found per sentence	1.45	4.66
average correct parse number*	1.10	2.45
average search focus to reach correct parse	19.60	24.48
average search focus in generating all parses	38.67	51.74
average time taken (seconds) to reach correct parse†	35.92	56.18
average time taken (seconds) in generating all parses†	81.63	125.94

SEARCH FOCUS RATIO TO CORRECT PARSE = 0.80
(with SPQR / without SPQR)

SEARCH FOCUS RATIO TO ALL PARSES = 0.75
(with SPQR / without SPQR)

TIMING RATIO TO CORRECT PARSE = 0.64

TIMING RATIO TO COMPLETION = 0.65

NEW CORRECT PARSES FOUND USING SPQR = 2

NEW CORRECT FIRST PARSES FOUND USING SPQR = 13

*That is, which parse, on the average, was the correct one.
†SPQR has not yet been optimized.

command] is anomalous for two reasons: The subject of ***engage*** cannot be an abstract concept such as ***loss***, and the object of ***engage*** must be a machine part.

4. EXPERIMENTAL RESULTS

The experimental results we present here are based on a sample of 31 sentences from one of our CASREP corpuses, each of which was parsed with and without invoking SPQR. We compare results obtained without using the selectional module to results obtained with the parser set to query the user about selectional patterns (starting from an empty pattern database). The chart in Table 2 summarizes the results for the 31 sentences.

One of the statistics presented in Table 2 is the SEARCH FOCUS, which is a measure of the efficiency of the parser in either reaching the correct parse of a sentence, or generating all possible parses. It is equal to the ratio of the number of nodes attached to the parse tree in the course of parsing (and possibly detached upon backtracking),[4] to the number of nodes in the completed, correct parse tree. Thus a search focus of 1.0 in reaching the correct parse would indicate that for every (branching) grammar rule tried, the first option was the correct one, or, in other words, that the parser had never backtracked.

The first line of the Table 2 chart deserves some explanation. One might wonder how a mechanism designed in part to ***rule out*** parses can actually ***produce*** a correct analysis for a sentence where none had been available without the module. The explanation is the COMMITTED DISJUNCTION mechanism we have implemented in our parser in order to reduce the (often spurious) ambiguity caused by allowing both full sentential and fragmentary readings. This pruning of the search space is most apparent when the parser is turned loose and set to generate all possible parses, as it was when we gathered the statistics summarized above. Recall that our parser tries full assertion parses before fragment parses. The effect of the COMMITTED DISJUNCTION mechanism is to commit the parser to produce only assertion parses (and no fragment parses) if an assertion parse is found. Fragment parses are tried only if no assertion parse is available. Thus no fragment reading will ever be generated for a sentence which can be analyzed as both an assertion and a fragment. This has proved to be the correct behavior in a majority of the texts we have analyzed. However, a fragment which can also be analyzed as an assertion will never receive a correct parse unless all assertion parses can be blocked using selection. Thus it is possible for selection to make available a correct syntactic analysis where none would be available without selection.

5. FUTURE PLANS

Our ultimate goal is to integrate SPQR with the domain model and the semantic component that maps syntactic constituents into predicates and associated thematic roles. At present, these components are developed independently. Our aim is to link these components in order to maintain consistency and facilitate updating the system. For example, if semantic rules exist to fill thematic roles of a given predicate, we should be able to ***derive*** a set of "surface" selectional patterns consistent with the underlying semantics. Similarly, given a set of selectional patterns, we should be able to suggest a (set of) semantic rule(s) consistent with the observed selection. In addition, if a word encountered in parsing is not represented in the domain model, it should be possible to suggest where the word should fit in the model, based on similarity to previously observed patterns. If, for example, in the CASREP domain, we encounter a sentence such as ***The widget broke***, but widgets do not appear in our domain model, the system would check for any patterns of the form [***X, break***]; if it finds such a pattern, e.g., [***machine_part, break***], the system can then suggest that ***widget*** be classified as a machine part in the domain model. If the user concurs, ***widget*** would then automatically be entered into the model.

In addition to the above work, which is already underway, we plan to improve the user interface, to measure the rate at which selectional patterns are acquired, and to investigate the use of selectional patterns in developing a weighting algorithm based on frequency of occurrence in the domain.

5.1. The User Interface

In the current implementation, the questions which the program asks the user are phrased in terms of grammatical categories, and are thus tailored to users who know what is meant by such terms as "SVO" and "noun-noun compounds". As a result, only linguists can be reasonably expected to make sense of the questions and provide meaningful answers. Our intended users, however, are not linguists, but rather domain experts who will know what can and cannot be said in the sublanguage, but who cannot be expected to reason in terms of grammatical categories. Deciding how to phrase questions designed to elicit the desired information is a difficult problem. Our first attempt will be to paraphrase the pattern. E.g., for the SVO pattern [***loss, install, sac***], the query to the user would be something like "Can a loss install a sac?".

[4] Another way to interpret this figure is that it represents the number of grammar rules tried

We are also examining what kind of knowledge the user must draw upon in order to answer the system's questions. Users' answers are usually based on a combination of commonsense knowledge (e.g., losses cannot install things) and domain-specific information. In certain cases, however, the user can be called upon to make fine linguistic distinctions. For example, in the sentence ***Sac disengaged immediately after alarm***, does the adverb ***immediately*** modify the verb ***disengaged***, or the prepositional phrase ***after alarm***? Most users, and even trained linguists familiar with the domain, find it difficult to provide definitive answers to such questions, because there is often no definitively correct answer. In this case, the adverbial attachment would seem to be genuinely ambiguous. It would be helpful to recognize patterns which a user cannot be reasonably expected to pass judgment on, and not generate queries about these, perhaps allowing them to succeed by default.

5.2. Measuring the System's Learning

As more sentences are parsed and more patterns are classified, we can expect the system to grow "smarter" in the sense that it will ask the user increasingly fewer questions. Eventually, the system should reach a state of reasonably complete domain knowledge, at which time few unknown patterns would be encountered, and the user would almost never be queried. We do not know how many sentences SPQR would have to examine before attaining such a plateau, but an estimate would be in the range of 500 to 1000 [Grishman1986]. We plan to measure the decrease in the frequency of queries to the user as a function of the number of sentences parsed and the number of patterns collected in order to evaluate the system's learning. This will enable us to determine the feasibility of using this technique to bootstrap into a new domain.

5.3. Preference-Based Parsing

A long-term goal is to implement a parsing algorithm based on preference rather than on the current success/failure paradigm. This would allow the system to use statistical information on the frequency of observed patterns as one factor in weighting. Frequently occurring patterns would be assigned greater weight than unknown patterns, and bad patterns would detract from the overall weighting. This would allow the system to make intelligent "guesses" about parsing without constantly querying the user.

ACKNOWLEDGMENTS

We would like to thank Marcia Linebarger for her helpful comments and insightful suggestions.

REFERENCES

[Ayuso1987]
Damaris M. Ayuso, Varda Shaked, and Ralph M. Weischedel, An Environment for Acquiring Semantic Information. In ***Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics***, Stanford, California, July 1987, pp. 32-40.

[Ballard1986]
Bruce W. Ballard and Douglas E. Stumberger, Semantic Acquisition in TELI: A Transportable, User-Customized Natural Language Processor. In ***Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics***, New York, NY, July 1986, pp. 20-29.

[Dahl1986]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT. In ***Proceedings of the Fifth National Conference on Artificial Intelligence***, Philadelphia, PA, 1986, pp. 1083-1088.

[Damerau1985]
Fred J. Damerau, Problems and Some Solutions in Customization of Natural Language Database Front Ends. ***ACM Transactions on Office Information Systems*** **3**(2), April 1985, pp. 165-184.

[Grishman1986]
Ralph Grishman, Lynette Hirschman, and Ngo Thanh Nhan, Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments. ***Computational Linguistics*** **12**(3), 1986, pp. 205-215.

[Grosz1983]
Barbara J. Grosz, TEAM: A Transportable Natural-Language Interface System. In ***Proceedings of the Conference on Applied Natural Language Processing***, Santa Monica, CA, February 1983, pp. 39-45.

[Hendrix1980]
Gary G. Hendrix, Mediating the Views of Databases and Database Users. In ***Proceedings of the Workshop on Data Abstraction, Databases, and Conceptual Modelling***, Pingree Park, CO, June 1980, pp. 131-132.

[Hirschman1982]
Lynette Hirschman and Karl Puder, Restriction Grammar in Prolog. In ***Proceedings of the First International Logic Programming Conference***, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseille, 1982, pp. 85-90.

[Hirschman1985]
Lynette Hirschman and Karl Puder, Restriction Grammar: A Prolog Implementation. In ***Logic Programming and its Applications***, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Hirschman1986a]
Lynette Hirschman, Discovering Sublanguage Structures. In ***Sublanguage: Description and Processing***, R. Kittredge and R. Grishman (ed.), Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.

[Hirschman1986b]
Lynette Hirschman, Conjunction in Meta-Restriction Grammar. ***Journal of Logic Programming*** **4**, 1986, pp. 299-328.

[Lang1987]
François-Michel Lang, A User's Guide to the Selection Module, Logic-Based Systems Technical Memo No. 68, Paoli Research Center, Unisys, Paoli, PA, October, 1987.

[Moser1984]
M. G. Moser, Domain Dependent Semantic Acquisition. In ***Proceedings of the First Conference on Artificial Intelligence Applications***, IEEE Computer Society, Denver, CO, December, 1984, pp. 13-18.

[Palmer1986]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information. In ***Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics***, Columbia University, New York, August 1986.

[Passonneau1986]
Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect, Logic-Based Systems Technical Memo No. 43, Paoli Research Center, Unisys, Paoli, PA, November, 1986.

[Sager1981]
Naomi Sager, ***Natural Language Information Processing: A Computer Grammar of English and Its Applications***. Addison-Wesley, Reading, Mass., 1981.

[Thompson1983]
Bozena H. Thompson and Frederick B. Thompson, Introducing ASK, A Simple Knowledgeable System. In ***Proceedings of the Conference on Applied Natural Language Processing***, Santa Monica, CA, February 1983, pp. 17-24.

[Thompson1985]
Bozena H. Thompson and Frederick B. Thompson, ASK is Transportable in Half a Dozen Ways. ***ACM Transactions on Office Information Systems*** **3**(2), April 1985, pp. 185-203.

[Tomita1984]
Masaru Tomita, Disambiguating Grammatically Ambiguous Sentences by Asking. In ***Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics***, Stanford, CA, July 1984, pp. 476-480.

In Proceedings of the 26th Annual Meeting of the Association of Computational Linguistics, 1988

SENTENCE FRAGMENTS REGULAR STRUCTURES

Marcia C. Linebarger, Deborah A. Dahl, Lynette Hirschman, Rebecca J. Passonneau

Paoli Research Center
Unisys Corporation
P.O. Box 517
Paoli, PA

ABSTRACT

This paper describes an analysis of telegraphic fragments as regular structures (not errors) handled by minimal extensions to a system designed for processing the standard language. The modular approach which has been implemented in the Unisys natural language processing system PUNDIT is based on a division of labor in which syntax regulates the occurrence and distribution of elided elements, and semantics and pragmatics use the system's standard mechanisms to interpret them.

1. INTRODUCTION

In this paper we discuss the syntactic, semantic, and pragmatic analysis of fragmentary sentences in English. Our central claim is that these sentences, which have often been classified in the literature with truly erroneous input such as misspellings (see, for example, the work discussed in [Kwasny1980, Thompson1980, Kwasny1981, Sondheimer1983, Eastman1981, Jensen1983]), are regular structures which can be processed by adding a small number of rules to the grammar and other components of the system. The syntactic regularity of fragment structures has been demonstrated elsewhere, notably in [Marsh1983, Hirschman1983]; we will focus here upon the regularity of these structures across all levels of linguistic representation. Because the syntactic component regularizes these structures into a form almost indistinguishable from full assertions, the semantic and pragmatic components are able to interpret them with few or no extensions to existing mechanisms. This process of incremental regularization of fragment structures is possible only within a linguistically modular system. Furthermore, we claim that although fragments may occur more frequently in specialized sublanguages than in the standard grammar, they do not provide evidence that sublanguages are based on grammatical principles fundamentally different from those underlying standard languages, as claimed by [Fitzpatrick1986], for example.

This paper is divided into five sections. The introductory section defines fragments and describes the scope of our work. In the second section, we consider certain properties of sentence fragments which motivate a modular approach. The third section describes our implementation of processing for fragments, to which each component of the system makes a distinct contribution. The fourth section describes the temporal analysis of fragments. Finally, the fifth section discusses the status of sublanguages characterized by these telegraphic constructions.

We define fragments as regular structures which are distinguished from full assertions by a missing element or elements which are normally syntactically obligatory. We distinguish them from errors on the basis of their regularity and consistency of interpretation, and because they appear to be generated intentionally. We are not denying the existence of true errors, nor that processing sentences containing true errors may require sophisticated techniques and deep reasoning. Rather, we are saying that fragments are distinct from errors, and can be handled in a quite general fashion, with minimal extensions to normal processing. Because we base the definition of *fragment* on the absence of a syntactically

*This work has been supported in part by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research; by National Science Foundation contract DCR-85-02205; and by Independent R&D funding from System Development Corporation, now part of Unisys Corporation. Approved for public release, distribution unlimited

obligatory element, noun phrases without articles are not considered to be fragmentary, since this omission is conditioned heavily by semantic factors such as the mass vs. count distinction. However, we have implemented a pragmatically based treatment of noun phrases without determiners, which is briefly discussed in Section 3.

Fragments, then, are defined here as elisions. We describe below the way in which these omissions are detected and subsequently 'filled in' by different modules of the system.

The problem of processing fragmentary sentences has arisen in the context of a large-scale natural language processing research project conducted at UNISYS over the past five years [Palmer1986, Hirschman1986, Dowding1987, Dahl1987]. We have developed a portable, broad-coverage text-processing system, PUNDIT.[1] Our initial applications have involved various message types, including: field engineering reports for maintenance of computers; Navy maintenance reports (Casualty Reports, or CASREPs) for starting air compressors; Navy intelligence reports (RAINFORMs); trouble and failure reports (TFRs) from Navy Vessels; and recently we have examined several medical domains (radiology reports, comments fields from a DNA sequence database). At least half the sentences in these corpora are fragments; Table 1 below gives a summary of the fragment content of three domains, showing the percent of centers which are classified as fragments. (Centers comprise all sentence types: assertions, questions, fragments, and so forth.)

Table 1. Fragments in three domains.

	Total centers	Percent fragments
CASREPS	153	53%
RAINFORM	41	78%
TFR	35	51%

The PUNDIT system is highly modular: it consists of a syntactic component, based on string grammar and restriction grammar [Sager1981, Hirschman1985]; a semantic component, based on inference-driven mapping, which decomposes predicating expressions into predicates and thematic roles [Palmer1983, Palmer1985]; and a pragmatics component which processes both referring expressions [Dahl1986], and temporal expressions [Passonneau1987, Passonneau1988].

[1] *Prolog UNDerstanding of Integrated Text*

2. DIVISION OF LABOR AMONG SYNTAX, SEMANTICS, AND PRAGMATICS

We argue here that sentence fragments provide a strong case for linguistically modular systems such as PUNDIT, because such elisions have distinct consequences at different levels of linguistic description. Our approach to fragments can be summarized by saying that syntax detects 'holes' in surface structure and creates dummy elements as placeholders for the missing elements; semantics and pragmatics interpret these placeholders at the appropriate point in sentence processing, utilizing the same mechanisms for fragments as for full assertions.

Syntax regulates the holes. Fragment elisions cannot be accounted for in purely semantic/pragmatic terms. This is evidenced by the fact that there are syntactic restrictions on omissions; the acceptability of a sentence fragment hinges on grammatical factors rather than, e.g., how readily the elided material can be inferred from context. For example, the discourse *Old house too small. *New one will be larger than _ was* (where the elided object of *than* is understood to be *old house*) is ill-formed, whereas a comparable discourse *First repairman ordered new air conditioner. Second repairman will install _* (where the elided object of *install* is understood to be *air conditioner*) is acceptable. In both cases above, the referent of the elided element is available from context, and yet only the second ellipsis sounds well-formed. Thus an appreciation of where such ellipses may occur is part of the *linguistic* knowledge of speakers of English and not simply a function of the contextual salience of elided elements. Since these restrictions concern structure rather than content, they would be difficult or impossible to state in a system such as a 'pure' semantic grammar which only recognized such omissions at the level of semantic/pragmatic representation.

Furthermore, it matters to semantics and pragmatics HOW an argument is omitted. The syntactic component must tell semantics whether a verb argument is missing because the verb is used intransitively (as in *The tiger was eating*, where the patient argument is not specified) or because of a fragment ellipsis (as in *Eaten by a tiger*, where the patient argument is missing because the subject of a passive sentence has been elided). Only in the latter case does the missing argument of *eat* function as an

antecedent subsequently in the discourse: compare *Eaten by a tiger. Had screamed bloody murder right before the attack* (where the victim and the screamer are the same) vs. *The tiger was eating. Had screamed bloody murder right before the attack* (where it is difficult or impossible to get the reading in which the victim and the screamer are the same).

Semantics and pragmatics fill the holes. In PUNDIT's treatment of fragments, each component contributes exactly what is appropriate to the specification of elided elements. Thus the syntax does not attempt to 'fill in' the holes that it discovers, unless that information is completely predictable given the structure at hand. Instead, it creates a dummy element. If the missing element is an elided subject, then the dummy element created by the syntactic component is assigned a referent by the pragmatics component. This referent is then assigned a thematic role by the semantics component like any other referent, and is subject to any selectional restrictions associated with the thematic role assigned to it. If the missing element is a verb, it is specified in either the syntactic or the semantic component, depending upon the fragment type.

3. PROCESSING FRAGMENTS IN PUNDIT

Although the initial PUNDIT system was designed to handle full, as opposed to fragmentary, sentences, one of the interesting results of our work is that it has required only very minor changes to the system to handle the basic fragment types introduced below. These included the additions of: 6 fragment BNF definitions to the grammar (a 5% increase in grammar size) and 7 context-sensitive restrictions (a 12% increase in the number of restrictions); one semantic rule for the interpretation of the dummy element inserted for missing verbs; a minor modification to the reference resolution mechanism to treat elided noun phrases like pronouns; and a small addition to the temporal processing mechanism to handle tenseless fragments. The small number of changes to the semantic and pragmatic components reflects the fact that these components are not 'aware' that they are interpreting fragmentary structures, because the regularization performed by the syntactic component renders them structurally indistinguishable from full assertions.

Fragments present parsing problems because the ellipsis creates degenerate structures. For example, a sequence such as *chest negative* can be analyzed as a 'zero-copula' fragment meaning *the chest X-ray is negative.* or a noun compound like *the negative of the chest.* This is compounded by the lack of derivational and inflectional morphology in English, so that in many cases it may not be possible to distinguish a noun from a verb (*repair parts*) or a past tense from a past participle (*decreased medication*). Adding fragment definitions to the grammar (especially if determiner omission is also allowed) results in an explosion of ambiguity. This problem has been noted and discussed by Kwasny and Sondheimer [Kwasny1981]. Their solution to the problem is to suggest special relaxation techniques for the analysis of fragments. However, in keeping with our thesis that fragments are normal constructions, we have chosen the alternative of constraining the explosion of parses in two ways. The first is the addition of a control structure to implement a limited form of preference via 'unbacktrackable' *or* (xor). This binary operator tries its second argument only if its first argument does not lead to a parse. In the grammar, this is used to prefer "the most structured" alternative. That is, full assertions are preferred over fragments -- if an assertion or other non-fragment parse is obtained, the parser does not try for a fragment parse.

The second mechanism that helps to control generation of incorrect parses is selection. PUNDIT applies surface selectional constraints incrementally, as the parse is built up [Lang1988]. For example, the phrase *air compressor* would NOT be allowed as a zerocopula because the construction *air is compressor* would fail selection.[2]

3.1. Fragment Types

The fragment types currently treated in PUNDIT include the following:

Zerocopula: a subject followed by a predicate, differing from a full clause only in the absence of a verb, as in *Impellor blade tip erosion evident*;

Tvo (tensed verb + object): a sentence missing its subject, as in *Believe the coupling from diesel to sac lube oil pump to be sheared*;

[2] Similarly, the assertion parse for the title of this paper would fail selection (sentences don't fragment structures), permitting the zerocopula fragment parse.

Nstg_frag: an isolated noun phrase (noun-string fragment), as in *Loss of oil pump pressure.*

Objbe_frag (object-of-be fragment): an isolated complement appropriate to the main verb *be*, as in *Unable to consistently start nr 1b gas turbine*;

Predicate: an isolated complement appropriate to auxiliary *be*, as in *Believed due to worn bushings*, where the full sentence counterpart is *Failure is believed (to be) due to worn bushings*;[3]

Obj_gap_fragment: a center (assertion, question, or other fragment structure) missing an obligatory noun phrase object, as in *Field engineer will replace _*

Note that we do not address here the processing of *response fragments* which occur in interactive discourse, typically as responses to questions.

The relative frequency of these six fragment types (expressed as a percentage of the total fragment content of each corpus) is summarised below.[4]

Table 2. Breakdown of fragments by type.

	CASREPS	RAINFORM	TFR
TVO	17.5%	40.6%	61%
ZC	52.5%	50%	16.6%
NF	25%	6.2%	16.6%
OBJBE	3.7%	0%	5.5%
PRED	1.2%	3.1%	0%
OBJ_GAP	0%	3.1%	0%

The processing of these basic fragment types can be summarized briefly as follows: a detailed surface parse tree is provided which represents the overt lexical content in its surface order. At this level, fragments bear very little resemblance to full assertions. But at the level of the *Intermediate Syntactic Representation* (ISR), which is a regularized representation of syntactic structure [Dahl1987..], fragments are regularized to parallel full assertions by the use of dummy elements standing in for the missing subject or verb. The CONTENT of these dummy elements, however, is left unspecified in most cases, to be filled in by the semantic or pragmatic components of the system.

Tvo. We consider first the **tvo**, a subjectless tensed clause such as *Operates normally*. This is parsed as a sequence of tensed verb and object: no subject is inferred at the level of surface structure. In the ISR, the missing subject is filled in by the dummy element elided. At the level of the ISR, then, the fragment *operates normally* differs from a full assertion such as *It operates normally* only by virtue of the element elided in place of an overt pronoun. The element elided is assigned a referent which subsequently fills a thematic role, exactly as if it were a pronoun; thus these two sentences get the same treatment from semantics and reference resolution[Dahl1986, Palmer1986].

Elided subjects in the domains we have looked at often refer to the writer of the report, so one strategy for interpreting them might be simply to assume that the filler of the elided subject is the writer of the report. This simple strategy is not sufficient in all cases. For example, in the CASREPS corpus we observe sequences such as the following, where the filler of the elided subject is provided by the previous sentence, and is clearly not the writer of the report.

(1) Problem appears to be caused by one or more of two hydraulic valves. Requires disassembly and investigation.

(2) Sac lube oil pressure decreases below alarm point approximately seven minutes after engagement. Believed due to worn bushings.

Thus, it is necessary to be able to treat elided subjects as pronouns in order to handle these sentences.

The effect of an elided subject on subsequent focusing is the same as that of an overt pronoun. We demonstrated in section 2 that elided subjects, but not semantically implicit arguments, are expected foci (or forward-looking centers [Grosz1986]) for later sentences.

[3] It is interesting to note that at least some of these types of fragments resemble non-fragmentary structures in other languages. **tvo** fragments, for example, can be compared to **zero-subject** sentences in Japanese, **serocopulas** resemble copular sentences in Arabic and Russian, and structures similar to **predicate** can be found in Cantonese (our thanks to K. Fu for the Cantonese data). This being the case, it is not surprising that analogous sentences in English can be processed without resorting to extragrammatical mechanisms.

[4] ZC = serocopula; NF = nstg_fragment; PRED = predicate; OBJBE = objbe_frag; OBJ_GAP = obj_gap_fragment

The basic assumption underlying this treatment is that the pragmatic analysis for elided subjects should be as similar to that of pronouns as possible. One piece of supporting evidence for this assumption is that in many languages, such as Japanese [Gundel1980, Hinds1983, Kameyama1985] the functional equivalent of unstressed pronouns in English is a zero, or elided noun phrase.[5] If zeros in other languages can correspond to unstressed pronouns in English, then we hypothesize that zeros in a sublanguage of English can correspond functionally to pronouns in standard English. In addition, since processing of pronouns is independently motivated, it is *a priori* simpler to try to fit elision into the pronominal paradigm, if possible, than to create an entirely separate component for handling elision. Under this hypothesis, then, two fragments represent simply a realization of a grammatical strategy that is generally available to languages of the world. [6]

Zerocopula. For a zerocopula (e.g., *Disk bad*), the surface parse tree rather than the ISR inserts a dummy verb, in order to enforce subcategorization constraints on the object. And in the ISR, this null verb is 'filled in' as the verb *be*. It is possible to fill in the verb at this level because no further semantic or pragmatic information is required in order to determine its content.[7] Hence the representation for *Disk bad* is nearly indistinguishable from that assigned to the corresponding *Disk is bad*; the only difference is in the absence of tense from the former. If the null verb represents auxiliary *be*, then, like an overt auxiliary, it does not appear in the regularized form. *Sac failing* thus receives a regularization with *fail* as the main verb. Thus the null verb inserted in the syntax is treated in the ISR in a fashion exactly parallel to the treatment of overt occurrences of *be*.

Nstg_frag. The syntactic parse tree for this fragment type contains no empty elements; it is a regular noun phrase, labeled as an **nstg_frag**. The ISR transforms it into a VSO sequence. This is done by treating it as the subject of an element **empty_verb**; in the semantic component, the subject of **empty_verb** is treated as the sole argument of a predicate existential(X). As a result, the **nstg_frag** *Failure of sac* and a synonymous assertion such as *Failure of sac occurred* are eventually mapped onto similar final representations by virtue of the temporal semantics of **empty_verb** and of the head of the noun phrase.

Objbe_frag and predicate. These are isolated complements; the same devices described above are utilized in their processing. The surface parse tree of these fragment types contains no empty elements; as with **serocopula**, the untensed verb *be* is inserted into the ISR; as with **tvo**, the dummy subject **elided** is also inserted in the ISR, to be filled in by reference resolution. Thus the simple adjective *Inoperative* will receive an ISR quite similar to that of *He/she/it is inoperative*.

Obj_gap_fragment. The final fragment type to be considered here is the elided noun phrase object. Such object elisions occur more widely in English in the context of instructions, as in *Handle _ with care*. Cookbooks are especially well-known respositories of elided objects, presumably because they are filled with instructions. Object elision also occurs in telegrammatic sublanguages generally, as in *Took _ under fire with missiles* from the Navy sighting messages. If these omissions occurred only in direct object position following the verb, one might argue for a lexical treatment; that is, such omissions could be treated as a lexical process of intransitivization rather than by explicitly representing gaps in the syntactic structure. However, noun phrase objects of prepositions may also be omitted, as in *Fragile. Do not tamper with _*. Thus we have chosen to represent such elisions with an explicit surface structure gap. This gap is permitted in most contexts where **nstgo** (noun phrase object) is found: as a direct object of the verb and as an object of a preposition. [8] In PUNDIT, elided objects are

[5] Stressed pronouns in English correspond to overt pronouns in languages like Japanese, as discussed in [Gundel1980, Gundel1981], and [Dahl1982].

[6] An interesting hypothesis, discussed by Gundel and Kameyama, is that the more topic prominent a language is, the more likely it is to have zero-NP's. Perhaps the fact that sublanguage messages are characterised by rigid, contextually supplied, topics contributes to the availability of the tvo fragment type in English.

[7] In some restricted subdomains, however, other verbs may be omitted: for example, in certain radiology reports an omitted verb may be interpreted as *show* rather than *be*. Hence we find *Chest films 1/10 little change*, paraphrasable as *Chest films show little change*

[8] Note, however, that there are some restrictions on the occurrence of these elements. They seem not to occur in

permitted only in a fragment type called obj_gap_fragment, which, like other fragment types, may be attempted only if an assertion parse has failed. Thus a sentence such as *Pressure was decreasing rapidly* will never be analyzed as containing an elided object, because there is a semantically acceptable assertion parse. In contrast, *John was decreasing gradually* will receive an elided object analysis, paraphrasable as *John was decreasing IT gradually*, because *John* is not an acceptable subject of intransitive *decrease*; only pressure or some equally measurable entity may be said to decrease. This selectional failure of the assertion parse permits the elided object analysis.

Our working hypothesis for determining the reference of object gaps is that they are, just like subject gaps, appropriately treated as pronouns. However, we have not as yet seen extensive data relevant to this hypothesis, and it remains subject to further testing.

These, then, are the fragment types currently implemented in PUNDIT. As mentioned above, we do not consider noun phrases without determiners to be fragments, because it is not clear that the missing element is *syntactically* obligatory. The interpretation of these noun phrases is treated as a pragmatic problem. In the style of speech characteristic of the CASREPs, determiners are nearly always omitted Their function must therefore be replaced by other mechanisms. One possible approach to this problem would be to have the system try to determine what the determiner would have been, had there been one, insert it, and then resume processing as if the determiner had been there all along. This approach was taken by [Marsh1981]. However, it was rejected here for two reasons. The first is that it was judged to be more error-prone than simply equipping the reference resolution component with the ability to handle noun phrases without determiners directly. [9] The second reason for not selecting this approach is that it would eliminate the distinction between noun phrases which originally had a determiner and those which did not. At some point in the development of the system it may become necessary to use this information.

The basic approach currently taken is to assume that the noun phrase is definite, that is, it triggers a search through the discourse context for a previously mentioned referent. If the search succeeds, the noun phrase is assumed to refer to that entity. If the search fails, a new discourse entity is created.

In summary, then, these fragment types are parsed 'as is' at the surface level; dummy elements are inserted into the ISR to bring fragments into close parallelism with full assertions. Because of the resulting structural similarity between these two sentence types, the semantic and pragmatic components can apply exactly the same interpretive processes to both fragments and assertions, using pre-existing mechanisms to 'fill in' the holes detected by syntax.

4. TEMPORAL ANALYSIS OF FRAGMENTS

Temporal processing of fragmentary sentences further supports the efficacy of a modular approach to the analysis of these strings.[10] In PUNDIT's current message domains, a single assumption leads to assignment of present or past tense in untensed fragments, depending on the aspectual properties of the fragment.[11] This assumption is that the messages report on actual situations which are of present relevance. Consequently, the default tense assignment is *present* unless tnis prevents assigning an actual time.[12]

For sentences having progressive grammatical aspect or stative lexical aspect, the assignment of present tense always permits interpreting

predicative objects, in double dative constructions, and, perhaps, in sentence adjuncts rather than arguments of the verb (Thus compare *Patient very ill. Do not operate on _* with *Operating room closed on Sunday. Do not perform surgery on_*.) One possibility is that these expressions can occur only where a definite pronoun would also be acceptable. In general, object gaps seem most acceptable where they represent an argument of a verb, either as direct object or as object of a preposition selected for by a verb.

[9] This ability would be required in any case, should the system be extended to process languages which do not have articles.

[10] For a discussion of the temporal component, cf. [Passonneau1987, Passonneau1988].

[11] Since the tvo fragment is tensed, its input to the time component is indistinguishable from that of a full sentence.

[12] Pundit does not currently take full advantage of modifier information that could indicate whether a situation has real time associated with it (e.g. **potential** *sac failure*), or whether a situation is past or present (e.g., *sac failure* **yesterday**, *pump* **now** *operating normally*)

a situation as having an actual time [Passonneau1987]. Thus, a present tense reading is always assigned to an untensed progressive fragment, such as *pressure decreasing*; or an untensed verbless copula with a non-participial complement, such as *pump inoperative*.

A non-progressive verbless copula fragment containing a cognitive state verb, as in *failure believed due to worn bushings*, is assigned a present tense reading. However, if the lexical verb has non-stative aspect,[13] e.g., *tests conducted* (process) or *new sac received* (transition event) then assignment of present tense conflicts with the assumption that the mentioned situation has occurred or is occurring. The simple present tense form of verbs in this class is given a habitual or iterative reading. That is, the corresponding full sentences in the present, *tests are conducted* and *new sac is received*, are interpreted as referring to types of situations that tend to occur, rather than to situations that have occurred. In order to permit actual temporal reference, these fragments are assigned a past tense reading.

Nstg_frag represents another case where present tense may conflict with lexical aspect. If an **nstg_frag** refers to a non-stative situation, the situation is interpreted as having an actual past time. This can be the case if the head of the noun phrase is a nominalization, and is derived from a verb in the process or transition event aspectual class. Thus, *investigation of problem* would be interpreted as an actual process which took place prior to the report time, and similarly, *sac failure* would be interpreted as a past transition event. On the other hand, an nstg_frag which refers to a stative situation, as in *inoperative pump*, is assigned present tense.

5. RELATION OF FRAGMENTS TO THE LARGER GRAMMAR

An important finding which has emerged from the investigation of sentence fragments in a variety of sublanguage domains is that the linguistic properties of these constructions are largely domain-independent. Assuming that these sentence fragments remain constant across different sublanguages, what is their relationship to the language at large? As indicated above, we believe that fragments should not be regarded as ERRORS, a position taken also by [Lehrberger1982, Marsh1983], and others. Fragments do occur with disproportionate frequency in some domains, such as field reports of mechanical failure or newspaper headlines. However, despite this frequency variation, it appears that the parser's preferences remain constant across domains. Therefore, even in telegraphic domains the preference is for a full assertion parse, if one is available. As discussed above, we have enforced this preference by means of the **xor** ('unbacktrackable' *or*) connective. Thus despite the greater frequency of fragments we do not require either a grammar or a preference structure different from that of standard English in order to apply the stable system grammar to these telegraphic messages.

Others have argued against this view of the relationship between sublanguages and the language at large. For example, Fitzpatrick et al. [Fitzpatrick1986] propose that fragments are subject to a constraint quite unlike any found in English generally. Their **Transitivity Constraint** (TC) requires that if a verb occurs as a transitive in a sublanguage with fragmentary messages, then it may not also occur in an intransitive form, even if the verb is ambiguous in the language at large. This constraint, they argue, provides evidence that sublanguage grammars have "a life of their own", since there is no such principle governing standard languages. The TC would also cut down on ambiguities arising out of object deletion, since a verb would be permitted to occur transitively or intransitively in a given subdomain, but not both.

As the authors recognize, this hypothesis runs into difficulty in the face of verbs such as *resume* (we find both *Sac resumed normal operation* and *Noise has resumed*), since *resume* occurs both transitively and intransitively in these cases. For these cases, the authors are forced to appeal to a problematic analysis of *resume* as syntactically transitive in both cases; they analyze *The noise has resumed*, for example, as deriving from a structure of the form *(Someone/something) resumed the noise*; that is, it is analyzed as underlyingly transitive. Other transitivity alternations which present potential counter-examples are treated as syntactic gapping processes. In fact, with these two mechanisms available, it is not clear what COULD provide a counter-example to

[13] Mourelatos' class of occurrences [Mourelatos1981]

the TC. The effect of all this insulation is to render the Transitivity Constraint vacuous. If all transitive/intransitive alternations can be treated as underlyingly transitive, then of course there will be no counter-examples to the transitivity constraint. Therefore we see no evidence that sublanguage grammars are subject to additional constraints of this nature.

In summary, this supports the view that fragmentary constructions in English are regular, grammatically constrained ellipses differing minimally from the standard language, rather than ill-formed, unpredictable sublanguage exotica. Within a modular system such as PUNDIT this regularity can be captured with the limited augmentations of the grammar described above.

ACKNOWLEDGMENTS

The system described in this paper has been developed by the entire natural language group at Unisys. In particular, we wish to acknowledge the contributions of John Dowding, who developed the ISR in conjunction with Deborah Dahl; and Martha Palmer's work on the semantics component. The ISR is based upon the work of Mark Gawron.

We thank Tim Finin and Martha Palmer as well as the anonymous reviewers for useful comments on an earlier version of this paper.

References

[Dahl1987]
Deborah A. Dahl, John Dowding, Lynette Hirschman, Francois Lang, Marcia Linebarger, Martha Palmer, Rebecca Passonneau, and Leslie Riley, Integrating Syntax, Semantics, and Discourse: DARPA Natural Language Understanding Program, R&D Status Report, Paoli Research Center, Unisys Defense Systems, May 14, 1987.

[Dahl1986]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[Dahl1982]
Deborah A. Dahl and Jeanette K. Gundel, Identifying Referents for two kinds of Pronouns. In *Minnesota Working Papers in Linguistics and Philosophy of Language*, Kathleen Houlihan (ed.), 1982, pp. 10-29.

[Dahl1987]
Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Passonneau, Nominalisations in PUNDIT, Proceedings of the 25th Annual Meeting of the ACL, Stanford, CA, July, 1987.

[Dowding1987]
John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar. In *Proc. of the 2nd International Workshop on Natural Language Understanding and Logic Programming*, Vancouver, B.C., Canada, 1987.

[Eastman1981]
C.M Eastman and D.S. McLean, On the Need for Parsing Ill-Formed Input. *American Journal of Computational Linguistics* **7**, 1981.

[Fitzpatrick1986]
E. Fitzpatrick, J. Bachenko, and D. Hindle, The Status of Telegraphic Sublanguages. In *Analyzing language in Restricted Domains*, R. Grishman and R. Kittredge (ed.), Lawrence Erlbaum Associates, Hillsdale, NY, 1986.

[Grosz1986]
Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein, Towards a Computational Theory of Discourse Interpretation, Mss., 1986.

[Gundel1981]
Jeanette K. Gundel and Deborah A. Dahl, The Comprehension of Focussed and Non-Focussed Pronouns, Proceedings of the Third Annual Meeting of the Cognitive Science Society, Berkeley, CA; August, 1981.

[Gundel1980]
Jeanette K. Gundel, Zero-NP Anaphora in Russian. *Chicago Linguistic Society Parasession on Pronouns and Anaphora*, 1980.

[Hinds1983]
John Hinds, Topic Continuity in Japanese. In *Topic Continuity in Discourse*, T. Givon (ed.), John Benjamins Publishing Company, Philadelphia, 1983.

[Hirschman1983]
Lynette Hirschman and Naomi Sager, Automatic Information Formatting of a Medical Sublanguage. In *Sublanguage: Studies of Language in Restricted Semantic Domains*, R. Kittredge and J. Lehrberger (ed.), Series of Foundations of Communications, Walter de Gruyter, Berlin, 1983, pp. 27-80.

[Hirschman1986]
L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming*(4), 1986, pp. 299-328.

[Hirschman1985]
L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[Jensen1983]
K. Jensen, G.E. Heidorn, L.A. Miller, and Y. Ravin, Parse Fitting and Prose Fixing: Getting a Hold on Ill-Formedness. *American Journal of Computational Linguistics* **9**, 1983.

[Kameyama1985]
Megumi Kameyama, Zero Anaphora: The Case of Japanese, Ph.D. thesis, Stanford University, 1985.

[Kwasny1981]
S.C. Kwasny and N.K. Sondheimer, Relaxation Techniques for Parsing Ill-Formed Input. *Am J. of Computational Linguistics* **7**, 1981, pp. 99-108.

[Kwasny1980]
Stan C. Kwasny, *Treatment of Ungrammatical and Extra-Grammatical Phenomena in Natural Language Understanding Systems*. Indiana University Linguistics Club, 1980.

[Lang1988]
Francois Lang and Lynette Hirschman, Improved Portability and Parsing Through Interactive Acquisition of Semantic Information, Proc. of the Second Conference on Applied Natural Language Processing, Austin, TX, February, 1988.

[Lehrberger1982]
J. Lehrberger, Automatic Translation and the Concept of Sublanguage. In *Sublanguage: Studies of Language in Restricted Semantic Domains*, R. Kittredge and J. Lehrberger (ed.), de Gruyter, Berlin, 1982.

[Marsh1983]
Elaine Marsh, Utilizing Domain-Specific Information for Processing Compact Text. In *Proceedings of the Conference on Applied Natural Language Processing*, Santa Monica, CA, February, 1983, pp. 99-103.

[Marsh1981]
Elaine Marsh, A or THE? Reconstruction of Omitted Articles in Medical Notes, Mss., 1981.

[Mourelatos1981]
Alexander P. D. Mourelatos, Events, Processes and States. In *Syntax and Semantics: Tense and Aspect*, P. J. Tedeschi and A. Zaenen (ed.), Academic Press, New York, 1981, pp. 191-212.

[Palmer1983]
M. Palmer, Inference Driven Semantic Analysis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, D.C., 1983.

[Palmer1986]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Presented at the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[Palmer1985]
Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[Passonneau1988]
Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics*, 1988.

[Passonneau1987]
Rebecca J. Passonneau, Situations and Intervals, Presented at the 25th Annual Meeting of the Association for Computational Linguistics, Stanford University, California, July 1987.

[Sager1981]
N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[Sondheimer1983]
N. K. Sondheimer and R. M. Weischedel, Meta-rules as a Basis for Processing Ill-Formed Input. *American Journal of Computational Linguistics* **9**(3-4), 1983.

[Thompson1980]
Bozena H. Thompson, Linguistic Analysis of Natural Language Communication with Computers. In *Proceedings of the 8th International Conference on Computational Linguistics*, Tokyo, 1980.

Passonneau, R. "Getting at Discourse Referents". *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, June, 1989.

GETTING AT DISCOURSE REFERENTS

Rebecca J. Passonneau
UNISYS, Paoli Research Center
P.O. Box 517, Paoli, PA 19301, USA

ABSTRACT

I examine how discourse anaphoric uses of the definite pronoun *it* contrast with similar uses of the demonstrative pronoun *that*. Their distinct contexts of use are characterized in terms of two contextual features—**persistence of grammatical subject** and **persistence of grammatical form**—which together demonstrate very clearly the interrelation among lexical choice, grammatical choices and the dimension of time in signalling the dynamic attentional state of a discourse.

1 Introduction

Languages vary in the number and kinds of grammatical distinctions encoded in their nominal and pronominal systems. Language-specific means for explicitly mentioning and re-mentioning discourse entities constrain what Grosz and Sidner refer to as the linguistic structure of discourse [2]. This in turn constrains the ways in which discourse participants can exploit linguistic structure for indicating or inferring *attentional state*. Attentional state, Grosz and Sidner's term for the dynamic representation of the participants' focus of attention [2], represents—among other things—which discourse entities are currently most salient. One function of attentional state is to help resolve pronominal references. English has a relatively impoverished set of definite pronouns in which gender is relevant only in the 3rd person singular, and where number—a fairly universal nominal category—is not relevant in the 2nd person. Yet even within the English pronominal system, there is a semantic contrast that provides language users with alternative means for accessing the same previously mentioned entities, therefore providing investigators of language with an opportunity to explore how distinct lexicogrammatical features correlate with distinct attentional processes. This is the contrast between demonstrative and non-demonstrative pronouns. In this paper I examine how certain uses of the singular definite pronoun *it* contrast with similar uses of the singular demonstrative pronoun *that*.

I present evidence that the two pronouns *it* and *that* have pragmatically distinct contexts of use which can be characterized in terms of a remarkably simple set of preconditions. First, in §2 I delineate the precise nature of the comparison carried out here. In §3.1, I describe the methods I used to collect and analyze a set of data drawn from ordinary conversational interactions. The outcome of a statistical analysis was a single, highly significant multi-dimensional distributional model, showing lexical choice to be reliably predicted by two features of the local context. The statistical results were strikingly clearcut, and provide confirmation that grammatical choices made by participants in a dialogue *prior to* a particular point in time correlate with lexical choice of either participant *at* that time.

Of over a dozen different variables that were examined, two alone turned out to have enormous predictive power in distinguishing between the typical contexts for the two pronouns. Very briefly, the first variable, **persistence of grammatical subject**, indicates whether both the antecedent and pronoun were subjects of their respective clauses. The second, **persistence of grammatical form**, indicates whether the antecedent was a single word phrase or a multi-word phrase, and if the latter, whether the phrase was syntactically more clause-like or more noun-like. Both variables point up the significance of the temporal dimension of discourse in two ways. The first has to do with the evanescence of surface syntactic form—the two features pertaining to the grammatical means used to refer to entities are relevant only for a short time, namely across two co-references [17]. The second has to do with the dual nature of referring expressions—as noted by Isard they are constrained by the prior context but immediately *alter the context and become part of it* [3] [18]. In §4 I discuss how the contrast between the definite and demonstrative pronouns is constrained by the local discourse context, and

how the constraining features of the local context in combination with the lexical contrast provides evidence about modelling the attentional state of discourse.

2 Comparability of *it* and *that*

Previous work has related the discourse deictic uses of *that* to the global segmental structure of discourse, and tied the contrast between *it* and *that* to the distinction between units of information introduced at the level of discourse segments versus units of information introduced at the level of the constituent structure of sentences [8] [12] [18]. This paper deals only with the latter category. That is, I am concerned with entities that are evoked into the discourse model by explicit mentions, i.e., noun phrases [19] or other intrasentential constituents, and with the difference between accessing these referents via the definite versus the demonstrative pronoun. Thus the data reported on here are restricted to cases where one of these pronouns has occurred with an explicit linguistic antecedent that is a syntactic argument.[1] A pronoun's antecedent was taken to be a prior linguistic expression evoking (or re-evoking) a discourse entity that provided a pronoun's referent. The two expressions were not constrained to be strictly coreferential since a wide variety of semantic relationships may hold between cospecifying expressions [1] [16] [19].

Syntactically *it* and *that* have very similar—though not identical—privileges of occurrence.[2] The following bullets briefly summarize their syntactic differences.

- *that*, but not *it*, is categorially ambiguous, occurring either as a determiner or as an independent pronoun
- *it*, but not *that*, has a reflexive and a possessive form (*itself*/**thatself*; *its*/**thats*)
- *it*, but not *that*, may occur in prepositional phrases where the pronoun in the PP corefers with a c-commanding NP (*the table with a drawer in it*/**that*)
- *it*, but not *that*, can be used non-referentially (*it*/**that is raining*; *it*/**that is hard to find an honest politician*)

These differences, though they may ultimately pertain to the phenomena presented here, won't be discussed further. In general, *that* can occur with the same syntactic types of antecedents with which *it* occurs. Thus, apart from prosodic differences—which were not considered here—the two pronouns are extremely comparable semantically as well as syntactically. Both pronouns are 3rd person, nonanimate, and singular. They are thus primarily distinguished by the semantic feature of demonstrativity.

An unforeseen but interesting fact is that the proximal demonstrative *this* occurred very rarely. So the relevant semantic contrast was that between definiteness and demonstrativity, and did not include the proximal/non-proximal contrast associated with *this* versus *that*. While I had originally planned to investigate the contrast between the two demonstrative pronominals as well, there were only 8 tokens of *this* out of ~700 pronouns whose antecedents were sentence internal arguments. This strongly suggests that however the attentional space of discourse entities is structured, it is not as differentiated as in the spatiotemporal domain, where the contrast between *this* and *that* is apparently more relevant. With respect to the contexts examined here, the proximal/nonproximal contrast between *this* and *that* is irrelevant.

A stretch of discourse evokes a set of discourse entities, some of which can be accessed pronominally. Of these, some can be accessed by *it*, and some can be accessed by *that*. The data I present suggest that the availability of focussed entities for definite and demonstrative pronominal reference differs, and that the consequences on the subsequent attentional state also differs. The conditions on and consequences of speaker choice of *it* or *that* must be pragmatic, and further, it is likely that the choice pertains to attentional state, since both pronominalization and demonstrativity play such a large role in indicating the attentional status of their referents (cf. [8], [15], [18]).

The following excerpts from my conversational data illustrate the syntactic variety of the pronouns' antecedents, and give a sense as well that substituting one pronoun for another sometimes results in an equally natural sounding discourse, with the difference being a very subtle one as

[1] Pronouns whose antecedents were independent tensed clauses or clausal conjuncts were excluded from consideration here. I reported on a much larger class of contexts in [12] [13] [14].

[2] The contexts which discriminate between them syntactically occurred very rarely in my data.

in 2.[3] Occasionally, the substitution creates discourse that is pragmatically odd, as in 6.

1. A: so [you plan to] work for a while, save some money, travel—B: save SOME MONEY and then blow IT (/THAT) off and then go to school

2. what does NOTORIETY mean to you, where does THAT (/IT) put you

3. I didn't really want TO (PAUSE) TEACH PEOPLE, THAT (/IT) wasn't the main focus

4. so in some ways, I'd like TO BE MY OWN BOSS, so THAT (/IT)'s something that in some way appeals to me very much

5. the drawback is THAT I'M ON CALL 24 HOURS A DAY but IT (/THAT) also means I get different periods of time off

6. I don't think EACH SITUATION IS INHERENTLY DIFFERENT FROM THE OTHER, at least, THAT (/IT)'s not the way I look at it

In this paper, I focus on the linguistic features of the local context, i.e., the context containing a pronoun token and its antecedent, in order to investigate the relationship between the pronominal features of demonstrativity and definiteness and the local attentional state of a discourse.

3 Statistical Analysis of the Conversational Data

3.1 Method

Psychologists and sociologists studying face-to-face interaction have argued that the baseline of interactive behavior is dyadic rather than monadic [4] [9]; similarly, in understanding how speakers cooperatively construct a discourse, the baseline behavior must be dialogic rather than monologic. The analytic methods employed here were adapted from those used in studying social interaction among individuals. I analyzed the local context of lexical choice between *it* and *that* in four career-counseling interviews. The interviews took place in a college career-counseling office, and were not staged. The final corpus consisted of over 3 1/2 hours of videotaped conversation between counselors and students. This provided an excellent source of data, with the speakers contributing tokens of *it/that* at the rate of roughly 1 in every 2 sentences, or a total of 1,183 tokens in all. Nearly all of these were indexed and coded for 16 contextual variables characterizing the linguistic structure of the local context.[4] These variables fell into two classes: those pertaining to the LINEAR ORGANIZATION OF DISCOURSE, or to the respective locations of the antecedent and pronoun,[5] and those pertaining to the SYNTACTIC FORM of the antecedent expression.

Statistical analysis was used as a discovery procedure for finding the strongest determinants of lexical choice, rather than to test a particular hypothesis. The goal was to find the best *fit* between the contextual variables and lexical choice, i.e., to include in a final statistical model only those variables which were highly predictive. I used log-linear statistical methods to construct a single best multi-dimensional model; log-linear analysis permits the use of the χ square statistic for greater than 2-dimensional tables. This is advantageous, because multi-dimensionality imposes more constraints on the statistical model, and is thus even more reliable than 2-dimensional tables in revealing non-chance correlations. In addition to multi-dimensionality, three other criteria guided the selection of the best *fit*: a statistically significant probability for the table, meaning a probability of 5.0% or lower; statistical independence of the predictive variables from one another, i.e., that they represented truly distinct phenomena, rather than overlapping factors; and finally, that the distributional patterns were the same for each individual speaker and for each separate conversation, in order to justify pooling the data into a single set.[6]

The antecedents of some of the pronouns occurred in the interlocutor's speech, but change of

[3] The relevant pronoun tokens and their antecedents appear in CAPS, and the substituted pronoun appears in parentheses to the right of the original. A: and B: are used to distinguish two speakers, where relevant. Text enclosed in brackets was added by the author to clarify the context.

[4] Certain repetitions, e.g., false starts, were excluded from consideration; cf. chapter 2 of [13]

[5] *Location* was construed very abstractly, and included, e.g., measures of whether the antecedent and pronoun were in the same, adjacent, or more distant sentences; how deeply embedded syntactically the antecedent and pronoun were; how many referential expressions with the same or conflicting semantic features of person, number and gender intervened between the pronoun and its antecedent; and their respective grammatical roles [13].

[6] The reliability of the data was tested by comparing within- and across-subjects statistical measures; i.e., I took into account the data for the conversations as a whole, each individual conversation, and each individual speaker [13].

Absolute Distributions				
Form of Ant't	Gram'l Roles	IT	THAT	Row Totals
Pronoun	Subj-Subj	147	31	178
	Other	110	54	164
NP	Subj-Subj	18	6	24
	Other	90	88	178
Non-NP Arg	Subj-Subj	3	3	6
	Other	25	66	91
Other	Subj-Subj	5	2	7
	Other	18	12	30
Column Totals		416	262	678

Main and Interaction Effects			
Source	Degrees of Freedom	χ-Square	Probability
Intercept	1	12.71	0.0004
Form of Antecedent	3	39.37	0.0001
Grammatical Roles	1	16.87	0.0001
Likelihood Ratio	3	0.35	0.9509

Table 1: A Multi-Dimensional Statistical Model of Lexical Choice

speaker within the local context had no effect on lexical choice, either alone, or in concert with other factors. Before pooling the data from all conversations and all individual speakers into a single population, the variability across conversations and speakers was tested and found to be insignificant. Thus the results presented below represent a speaker behavior—lexical choice of pronoun—that is extraordinarily consistent across speakers, that is independent of whether a pronoun and its antecedent occurred in the same speaker's turn, independent of individual speaker and even of individual conversation. Consequently, it is justifiable to assume that the factors found to predict lexical choice pertain to communicatively relevant purposes. In other words, whatever these factors are, they presumably pertain not only to models of speech production, but also to models of speech comprehension.

3.2 Results

Table 1 gives the distribution of pronouns across the relevant contexts and gives the probabilities and χ-squares for the two contextual variables and their intercept, i.e., the interaction between them.[7] The very low probability of 0.04% for

[7]Note that the 4th category of Antecedent—**Other**—includes a mixture of atypical arguments, primarily adverbial in nature, like the adverbial argument of *go* in *go far*.

Form of Antecedent	Subsequent Pronoun: Subject IT	Subject THAT	Non-Subject IT	Non-Subject THAT
Pronominal Subject	147 96.0 **27.1**	31 48.7 **6.4**	39 48.7 1.9	19 42.4 **12.9**
Pronominal Non-Subject	37 43.1 .9	21 21.9 .0	34 21.9 **6.7**	14 19.1 1.3
NP Subject	18 18.3 .0	6 9.3 1.1	11 9.3 .3	10 8.1 .1
NP Non-Subject	43 63.9 **6.8**	33 32.4 .0	36 32.4 .4	45 28.2 **10.0**
Non-NP Subject	8 6.1 .6	5 3.1 1.2	1 3.1 1.4	1 2.7 1.1
Non-NP Non-Subject	23 48.4 **13.3**	44 24.6 **15.3**	19 24.6 1.3	33 21.4 **6.3**
Table χ-Square				116.3
Degrees of Freedom				7
Probability				0.001

Table 2: A Two-Way Distributional View of the Data, showing Absolute Frequency, Expected Frequency, and χ-squares for each Cell

the intercept indicates that the two variables are clearly independent, or in other words, represent two distinct contexts. The exceedingly low probabilities of 0.01% for the contextual variables and the highly significant table χ-square (i.e., close to 1) indicate that the model is extremely significant.[8] The correlation between the dependent dimension of lexical choice and the two independent dimensions, **persistence of grammatical subject** and **persistence of grammatical form**, presents an intuitively very satisfying view—yet not an obvious one a priori—of how all three variables conspire together to convey the current attentional status of a discourse referent. First I will summarize the effects of the two contextual variables one at a time. Then I will review the distributionally significant facts as a whole.

First Dimension: Persistence of Grammatical Subject. The first dimension of the model is binary and the two contexts it defines are in diametric opposition to one another; *it* was likely

[8]The cutoff is generally 5%. 1% is deemed to be very significant

to occur in exactly one of the two contexts, and *that* was likely to occur in the opposing context. If both referring expressions were subjects, then the lexical choice was far more likely to be *it* than *that*. All it took for the balance to swing in favor of the demonstrative was for either the pronoun itself or for its antecedent to be a non-subject. The two relevant contexts, then are:

- those in which both the antecedent and the target pronoun are syntactic subjects; ⇒ IT
- all other contexts. ⇒ THAT

Parallelism has sometimes been suggested as an organizing factor across clauses. It is certainly a strong stylistic device, but did not make a strong enough independent contribution to the statistical model to be included as a distinct variable. To repeat, the crucial factor was found to be that both expressions were subjects, not that both had the same grammatical function in their respective clauses. In §4.1 I will review the relationship of these results to the centering literature [1] [5] [6].

Second Dimension: Persistence of Grammatical Form. While many grammatical distinctions among sentence constituents are possible, the syntactic form of a pronoun's antecedent correlated with the choice between *it* and *that* in the following very specific way. The 3 discriminating contexts were where the antecedent was:

- any pronoun—the lexical choice for an antecedent pronoun had no effect on the lexical choice of the subsequent pronoun; ⇒ IT
- a canonical NP headed by a noun (including nominalizations); ⇒ IT or THAT
- and all other types of constituents. ⇒ THAT

The latter category included gerundives, infinitival expressions, and embedded finite clauses.[9] For contexts with a pronominal antecedent, the lexical choice was far more likely to be *it*. For canonical NP antecedents, *it* and *that* were equally likely, regardless of the type of head. For other types of constituents, *that* was far more likely. Thus there are two opposing contexts and one which doesn't discriminate between the two pronouns, i.e., a context in which the opposition is neutralized.

[9] Cf. [14] for a detailed discussion of how the precise dividing line between types of antecedents was determined

The dynamic component of this dimension is that it indicates, for a consecutive pair of co-specifying expressions, whether there has been a shift towards a surface form that is syntactically more compact and semantically less explicit, and if so, how great a shift. In the first context, where the antecedent is already pronominal, there is no shift, and *it* has a much higher probability of occurrence than *that*. The context in which there is a shift from a lexical NP to a phrasal NP, i.e., a shift from a reduced form to an unreduced one, but no categorial shift, doesn't discriminate between the two pronouns. The context favoring *that* is the one in which there is not only a shift from a single word to a multi-word phrase, but also a change in the categorial status of the phrase from a non-NP constituent to a lexical NP.

Full 3-way model. Table 2 displays the data in a finer-grained two-dimensional χ-square table in order to show separately all 4 of the possible outcomes, i.e., *it* or *that* as a subject or non-subject. In this table, the row headings represent the antecedent's form and grammatical role; the column headings represent the lexical choice and grammatical role of the subsequent pronominal expression. Each cell of the table indicates the absolute frequency, the expected frequency given a non-chance distribution, and the cell χ-square, with the latter in boldface type to indicate the significant cells. This is a somewhat more perspicuous view of the data because it can be displayed schematically in terms of initial states, final states, and enhanced, suppressed or neutral transitions, as in Fig. 1. However, it is also a somewhat misleading transformation of the 3-dimensional view given in table 1, because it suggests that the grammatical role of a pronoun and that of its antecedent are independent factors. Since the statistical model shown in table 1 is actually the best *fit* of the data, better than other models that were tested in which the grammatical role of each expression was treated separately [13], it is crucial to recognize that the statistically significant factor is the pair-wise comparison of subject status.

Large cell χ-squares in Table 2 indicate the significant contexts, and a comparison of the absolute and expected frequencies in these cells indicate whether the context is significantly frequent or significantly infrequent. Thus there are 3 types of cells in the table representing the contexts of lexical choice as chance events, as enhanced events, or as suppressed events. In Fig. 1, I have translated

1.	**Pro-Subj**	⊢	IT-Subj
2.		⊣	THAT-Subj
3.			IT-NonSubj
4.		⊣	THAT-NonSubj
5.	Pro-NonSubj		IT-Subj
6.			THAT-Subj
7.		⊢	IT-NonSubj
8.			THAT-NonSubj
9.	NP-Subj		IT-Subj
10.			THAT-Subj
11.			IT-NonSubj
12.			THAT-NonSubj
13.	**NP-NonSubj**	⊣	IT-Subj
14			THAT-Subj
15.			IT-NonSubj
16.		⊢	THAT-NonSubj
17.	NonNP-Subj		
18.			
19.			
20.			
21.	**NonNP-NonSubj**	⊣	IT-Subj
22.		⊢	THAT-Subj
23.			IT-NonSubj
24.		⊢	THAT-NonSubj

Figure 1: Schematic Representation of Table 2 as a set of State Transitions

the table into a set of 3 types of state transitions. Initial states are in the left column and final states in the right one. The 3 types of transition are one which is unaffected by the contrast between *it* and *that* (no symbol), one which is enhanced (⊢), and one which is suppressed (⊣). The initial states in boldface indicate for each antecedent type which of the two grammatical role states was more likely, subject or non-subject. Absence of final states for the *nonNP-Subj* initial state indicates that this set of contexts is extremely rare. In the following section, I discuss the relation of these events to an abstract model of attentional state.

4 Discussion

The outcome of this study is not a model of attentional processes *per se*, but rather, a set of factors pertaining to attentional structure that elucidates the shifting functions of the demonstrative pronoun in English discourse. The particular function served by *that* seems to depend on what functional contrasts are available given the current attentional state.

It is most useful to think of the data in terms of two major categories of phenomena. The first category is where a discourse entity has already been mentioned pronominally. In this case, maintenance of reference in subject grammatical role is a particularly signficant determinant of the choice between *it* and *that*. This effect is discussed in §4.1 in relation to the notion of centering. The second category is where a discourse entity most recently evoked by a multi-word phrase is subsequently referenced by a pronoun. While grammatical role is relevant here, its relevance seems to depend on a more salient distinction pertaining to the syntactico-semantic type of the discourse entity, as discussed in §4.2.

4.1 Definite/Demonstrative Pronouns and Centering

The literature on attentional state has shown that both pronominalization and grammatical role affect the attentional status of a discourse entity. In this section I will show how the use of the definite pronoun *it* conforms in particular to the predictions made by Kameyama [6] [5] regarding canonical and non-canonical center-retention, and that the demonstrative pronoun is incompatible with center-retention.

The centering model predicts that an utterance will contain a referent that is distinguished as the backward looking center (Cb) [1], and that if the Cb of an utterance is coreferential with the Cb of the prior utterance, it will be pronominalized [1]. Kameyama [6] proposes that there are two means for retaining a discourse entity as the Cb, canonical center-retention—both references in subject role—and non-canonical center retention—neither reference in subject role. As shown in Fig. 1, the most enhanced context for lexical choice of *it* (context 1) was where both the pronoun and its pronominal antecedent were subjects, i.e., canonical center-retention. The next most enhanced context for *it* (context 7) was where neither the pronoun nor its pronominal antecedent were subjects, i.e., non-canonical center-retention. Thus, the definite pronoun correlates with both canonical and non-canonical center retention.

Lexical choice of *it* is actively suppressed in contexts which are incompatible with center-retention. Note in Fig. 1 that if the antecedent is neither a pronoun nor a subject, a subsequent reference via *it* in subject role is suppressed (contexts 13 and 21). The only (non-rare) context where an *it* subject is neither enhanced nor suppressed is

where the antecedent is a canonical NP in subject role (context 9) (cf. §4.2).

The demonstrative pronoun is actively suppressed in the case of canonical center-retention (context 2); i.e, given two successive pronominal references to the same entity where reference is maintained in subject role, the referent's attentional state is such that it precludes demonstrative reference. Use of *that* is also suppressed if the antecedent is a potential candidate for canonical center retention, even if reference is not maintained in subject role (context 4).

Attentional state is only one component of a discourse structure. The discourse model as a whole will contain representations of many of the things to whcih the discourse participants can subsequently refer, including discourse entities evoked by NPs, and additionally, as argued by Webber [18], discourse segments. Webber notes that discourse segment referents may have a different status from the discourse entities evoked by NPs, at least until they have been pronominally referenced. However, she suggests that when a demonstrative pronoun refers to a discourse entity, it accesses that entity by a process which first involves accessing the discourse segment in which the discourse entity is introduced. In other words, she posits two distinct referential processes, deictic and anaphoric reference, and suggests that even when a demonstrative pronoun refers to a discourse entity, the process of finding the referent is distinct in kind from anaphoric reference to the same entity. While I have no evidence that bears directly on such a claim, my data do indicate that some entities in a discourse segment are ordinarily unavailable via the demonstrative pronoun, namely entities that would be expected canonical centers, as described in the preceding paragraph. Thus my data support the view that there are distinct processes for accessing entities in the model.

It is relevant to note here that the notion of Cb is generally discussed in terms of links between successive utterances. Since there is an extraordinary frequency of conjoined sentences in conversational language, I distinguished between utterances and independent clauses within an utterance. The successive references in my data were in successive sentences a majority of the time (roughly 2/3; cf. [13]), but were sometimes separated by one or more sentences (roughly 1/6) and sometimes occurred in the same sentence (roughly 1/6). This distance factor had no correlation with lexical choice of pronoun, which suggests that discourse segment structure interacts with centering. The relevant local context for center-retention may not be successive sentences/utterances, but rather, successive sentences/utterances within the same discourse segment. In any case, for the data presented here, the relevant local context consisted of two successive co-specifying phrases, not two successive utterances.

Since the primary objective of this study was to examine various features of the context immediately preceding a given type of pronoun, rather than to track the discourse history of particular entities, little can be said here about the general case of multiple successive references to the same entity. However, I did investigate a subset of this general case, namely, successive pronominal references to the same entity where the initial mention was a canonical NP, and where each next co-specifying pronoun served as the antecedent for a subsequent pronoun. I refer to these as pronoun chains.[10] The relative likelihood of *it* and *that* was the same for the first slot in the chain, which conforms to the general distribution for pronouns with NP antecedents. The ratio of *it* to *that* in the last position of a chain conforms to chance, i.e., it equals the ratio of *it* to *that* in the pronoun chain sample. But within a chain, *that* is strongly predicted by ***persistence of grammatical form***. The demonstrative occured rarely within chains, but where it did occur, either the demonstrative token or its antecedent was a non-subject. This was found to be the only factor pertaining to linguistic structure that affected the occurrence of *that* within a pronoun chain.

A final set of conclusions derived from the pronominal initial states in Fig. 1 pertains to the non-predictive contexts, i.e., those which neither enhance nor suppress center-retention, and those which neither enhance nor suppress demonstrative reference. These are cases where there is either a shift in grammatical role, or where the lexical choice is *that* (contexts 3, 5, 6 and 8). When a center is not retained across two successive utterances (in the same discourse segement), then it is likely that the global context is affected [1], perhaps by a center-shift (cf. [5]), or by a segment boundary (cf. [7], [11]). Centers seem generally to be unavailable for demonstrative reference, but contexts 6 and 8

[10] The term seems to have appeared in the philosophical and linguistic literature at about the same time, e.g., in works by K. Donnellan, C. Chastain, M. Halliday and D. Zubin. There were a total of 101 such chains comprising 305 total pronoun tokens; they ranged in length from 2 to 13 pronouns.

in Fig. 1 perhaps represent a mechanism whereby an entity maintained as center can become available for demonstrative reference; e.g., context 6 may coincide with the *chaining* context discussed in the preceding paragraph, whereby a locally focussed entity can be accessed by *that* just in case the prior reference was a non-subject. Context 8 suggests that demonstrative reference is more available in contexts of non-canonical center retention than canonical center retention.

4.2 Non-Centered Discourse Entities

I have argued elsewhere that the crucial distinction for the category of non-pronominal antecedents is the contrast between true NPs with NP syntax, versus all other types of syntactic arguments ([12] [14]). This raises two important issues pertaining to the status of the discourse entities evoked by NPs versus other kinds of arguments. The first is that if non-NP arguments evoke discourse entities, which they certainly must, such entities apparently have a different status in the model than discourse entities evoked by NPs, given that the combination of lexical choice between *it* and *that* and grammatical function so clearly distinguish them. The second issue is that although the difference in status seems—at first blush—to correlate with a syntactic property, the distinction may ultimately be semantic in nature. I will discuss each issue in turn.

Two of the non-pronominal initial states in Fig. 1 are distinguished by neither enhancing nor suppressing any of the possible transitions to *it* or *that*: NP subjects (9-12), and non-NP non-subjects (17-20). The extreme rarity of the latter suggests that non-NPs don't occur as grammatical subjects, or that when they do, they are not likely to be re-evoked by a pronoun. On the other hand, NP subjects are fairly frequent in the contexts where *it* or *that* occurs with a non-pronominal antecedent, thus the absence here of enhanced or suppressed transitions suggests that an entity mentioned as an NP subject is free to be accessed in a variety of ways, or more precisely, that it has a relatively unspecified attentional state. It is neither a particularly likely Cb nor is it particularly available or unavailable for demonstrative reference.

The two remaining non-subject initial states, i.e., NP non-subjects and non-NP non-subjects, both suppress subsequent reference via *it* subjects, as mentioned in the previous section. While NP subjects apparently have a somewhat unspecified attentional status, NP non-subjects enhance the lexical choice of non-subject *that*. It appears that discourse entities evoked by NPs which are not subjects are in an attentional state that is quite different from that of canonical center retention.

It is especially interesting that when the antecedent is a non-NP non-subject, a subsequent pronominal reference is most likely to be demonstrative, and most likely to be a subject.[11] The enhancement of a *that*-subject context is completely contrary to the pattern established for subjects and for the demonstrative pronoun. These facts contribute to the view that entities evoked by non-NP constituents have a special status, but what this status is remains to be determined. In previous work, I emphasized the syntactic distinction with respect to lexical choice between *it* and *that* [14]. Although the most obvious difference is the purely syntactic one, the syntactic distinction between NP and non-NP constituents has a number of semantico-pragmatic consequences. In discussing the nominal and temporal anaphora within Kamp's framework of discourse representation structures (DRS), Partee raised the question of the difference in status between event-*describing* clauses and nominalizations [10]. Independent clauses differ from the class of non-NP constituents under consideration here in that the latter occur as arguments of superordinate verbs, and are thus entities participating in a described situation, as well as descriptions of situations. However, true noun phrases—whether they describe events or not—can have definite or indefinite determiners, and cannot have tense or any aspectual categories associated with the verb. The study presented here brings us no closer to a solution to the questions posed by Partee regarding the ontology and representation of different kinds of event descriptions, but it does offer further confirmation that entities evoked by NP and non-NP constituents have a different conceptual status, given the different possibilities for lexical choice and grammatical role of a subsequent pronominal mention.

5 Conclusion

The following bullets encapsulate the observations made in §4:

[11] Cf. examples 3-6 in §2 for illustrations.

- Lexical choice of *it* indicates canonical or non-canonical center retention
- Lexical choice of *it* in subject role conflicts with non-subject antecedents, but is compatible with an NP-subject antecedent
- Lexical choice of *that* blocks canonical center retention
- Lexical choice of *that* may be more compatible with non-canonical center retention
- Lexical choice of *that* in subject role is most likely when the antecedent is a non-NP constituent
- Lexical choice of *that* is enhanced when the antecedent is a non-N$_1$ constituent
- Lexical choice of *that* is enhanced when the antecedent NP is a non-subject
- NP subjects have a relatively unspecified attentional status

ACKNOWLEDGEMENTS

The data collection and statistical analysis were supported by Sloan Foundation Grant 1-5680-22-4898. The computational analysis and preparation of the paper were supported by DARPA Contract N00014-85-C-0012. Many thanks to Elena Levy, Deborah Dahl, Megumi Kameyama, Carl Weir, Bonnie Webber and David Searls for helpful discussion, commentary and criticism.

Bibliography

[1] B. J. Grosz, A. K. Joshi, and S. Weinstein. Providing a unified account of definite noun phrases in discourse. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 44–50, 1983.

[2] B. J. Grosz and C. L. Sidner. Attention, intentions and the structure of discourse. *Computational Linguistics*, 175–204, 1986.

[3] S. Isard. Changing the context. In E.L.Keenan, editor, *Formal Semantics of Natural Language*, pages 287–296, Cambridge U. Press, Cambridge, 1975.

[4] S. Duncan Jr., B. Kanki, H. Mokros, and D. Fiske. Pseudounilaterality, simple-rate variables, and other ills to which interaction research is heir. *Journal of Personality and Social Psychology*, 1335–1348, 1984.

[5] M. Kameyama. Computing japanese discourse: grammatical disambiguation with centering constraints. In *Proceedings of University of Manchester Institute of Science and Technology: Workshop on Computing Japanese*, 1987.

[6] M. Kameyama. A property-sharing constraint in centering. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 200–206, 1986.

[7] E. Levy. *Communicating Thematic Structure in Narrative Discourse: The Use of Referring Terms and Gestures*. PhD thesis, University of Chicago, 1984.

[8] C. Linde. Focus of attention and the choice of pronouns in discourse. In Talmy Givon, editor, *Syntax and Semantics: Discourse and Syntax*, pages 337–354, Academic Press, New York, 1979.

[9] H. B. Mokros. *Patterns of Persistence and Change in the Sequencing of Nonverbal Actions*. PhD thesis, University of Chicago, 1984.

[10] B. H. Partee. Nominal and temporal anaphora. *Linguistics and Philosophy*, 243–286, 1984.

[11] R. Reichman. *Getting Computers to Talk Like You and Me*. MIT Press, Cambridge, Massachusetts, 1985.

[12] R. J. (Passonneau) Schiffman. Categories of discourse deixis. 1984. Presented at the 29th Annual Conference of the International Linguistics Association.

[13] R. J. (Passonneau) Schiffman. *Discourse Constraints on it and that: A Study of Language Use in Career-Counseling Interviews*. PhD thesis, University of Chicago, 1985.

[14] R. J. (Passonneau) Schiffman. The two nominal anaphors *it* and *that*. In *Proceedings of the 20th Regional Meeting of the Chicago Linguistic Society*, pages 322–357, 1984.

[15] C. L. Sidner. Focusing in the comprehension of definite anaphora. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*, pages 267–330, The MIT Press, Cambridge, Massachusetts, 1983.

[16] C. L. Sidner. *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*. Technical Report, MIT AI Lab, 1979.

[17] M. Silverstein. Cognitive implications of a referential hierarchy. 1980. Unpublished ms.

[18] B. L. Webber. Discourse deixis: reference to discourse segments. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 113–122, 1988.

[19] B. L. Webber. So what can we talk about now. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*, pages 331–372, The MIT Press, Cambridge, Massachusetts, 1983.

Passonneau, R. "A Computational Model of the Semantics of Tense and Aspect". *Computational Linguistics*, vol. 14, no. 2, June, 1988.

A COMPUTATIONAL MODEL OF THE SEMANTICS OF TENSE AND ASPECT[1]

Rebecca J. Passonneau
Paoli Research Center, Defense Systems, UNISYS[2]

Abstract

The PUNDIT natural language system processes references to situations and the intervals over which they hold using an algorithm that integrates the analysis of tense and aspect. For each tensed clause, PUNDIT processes the main verb and its grammatical categories of tense, perfect and progressive in order to extract three complementary pieces of temporal information. The first is whether a situation has ***actual*** time associated with it. Secondly, for each situation which is presumed to take place in actual time, PUNDIT represents its ***temporal structure*** as one of three situation types: a state, process or transition event. The temporal structures of each of these situation types consist of one or more intervals. The intervals are characterized by two features: kinesis, which pertains to their internal structure, and boundedness, which constrains the manner in which they get located in time. Thirdly, the computation of ***temporal location*** exploits the three temporal indices proposed by Reichenbach: ***Event Time***, ***Speech Time*** and ***Reference Time***. Here, however, Event Time is formulated as a single component of the full temporal structure of a situation in order to provide an integrated treatment of tense and aspect.

1. Introduction

The PUNDIT text-processing system extracts temporal information about real-world situations from short message texts.[3] This involves three complementary analyses. First, PUNDIT determines whether a situation has ***actual time*** associated with it. A reference to a possible or potential situation, for example, would need a different treatment. Second, it determines the ***temporal structure*** of the predicated situation, or the manner in which it evolves through time. Finally, it analyzes the ***temporal location*** of the actual situations with respect to the time of text production or to the times of other situations. These three pieces of information are derived from the lexical head of a predication (verbal, adjectival or nominal), its grammatical inflections (tense, progressive, perfect), and finally, temporal adverbs such as ***before***, ***after*** and ***when***. Each of these components of temporal meaning is assigned a context-dependent compositional

[1]This work was supported by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research. APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

[2]Formerly Paoli Research Center, SDC—A Burroughs Company.

[3]Prolog UNDerstanding of Integrated Text. It is a modular system, implemented in Quintus Prolog, with distinct syntactic, semantic and pragmatic components [Dahl87a, Dahl86, Dowding87, Palmer85].

semantics. A fundamental premise of this approach is that the several sentence elements contributing temporal information can and should be analyzed in tandem [Mourelatos81, Dowty86] in order to determine the times for which predications are asserted to hold. This is accomplished by means of a model of the semantics of time which incorporates both aspect and a Reichenbachian treatment of tense [Reichenbach47].

The temporal analysis component described here was originally designed to handle PUNDIT's first text domain, CASREP messages, which are reports describing equipment failures on navy ships.[4] This domain was a particularly appropriate one for implementing a component to analyze the time information contained explicitly within the individual sentences of a text. CASREPs are diagnostic reports consisting of simple declarative sentences. They present a cumulative description of the current status of a particular piece of equipment rather than narrating a sequence of events. Within one sentence, several different situations may be mentioned, linked together by explicit temporal connectives such as ***before*** and ***after***. It is thus possible to extract a good deal of the important temporal information from these texts without handling inter-sentential temporal relations. However, the implementation of the temporal semantic component described here lays the necessary groundwork for eventually computing inter-sentential relations along lines Webber proposes in [Webber87] and this volume.[5] The capacity to process inter-sentential temporal relations is, of course, essential for adequately handling narrative data

2. Temporal Information

The premise of the present work is that accurate computation of the temporal semantics of the verb and its grammatical categories of tense, perfect and progressive provides a foundation for computing other kinds of temporal information, including the interpretion of temporal adverbials.[6] However, the task of modeling the semantic contribution of the verb and its categories is a complex one because temporal information is distributed across several non-univocal lexical and grammatical elements. As the extensive linguistic and philosophical literature on tense and aspect demonstrates, the precise temporal contribution of any one surface category of the verb is contingent upon co-occurring verbal categories, as well as upon the inherent meaning of the verb, and even the nature of the verb's arguments [Comrie76, Dowty79, Mourelatos81, Vlach81, Vendler67]. Hence, even a

[4] PUNDIT has now been adapted to four domains.

[5] Webber, in work carried out in part at the Paoli Research Center, proposes a focusing algorithm for computing inter-sentential temporal relations which is analogous to Sidner's focusing mechanism for definite anaphoric expressions. Future work by Webber and Passonneau will integrate the two dimensions of inter- and intra-sentential temporal analysis.

[6] Various types of tenseless predications are processed by PUNDIT's temporal component, including nominalizations, certain clausal modifiers of noun phrases (e.g., ***pressure decreasing below 60 psig caused the pump to fail***), and sentence fragments[Linebarger88]. However, this paper focuses on the simpler case of tensed clauses.

preliminary solution to the computational problems of interpreting temporal information in natural language requires recognizing the relevant semantic interdependencies. This paper proposes a solution to the computational task of extracting temporal information from simple declarative sentences based on separating temporal analysis into distinct tasks, each of which has access to a selected portion of the temporal input. The ultimate goal is to represent temporal information as explicitly as possible at each stage of analysis in order to provide the appropriate information for the next stage. Because the representations are constructed incrementally, it is important that they should be explicit about what has been derived so far, yet sufficiently non-committal to avoid conflicting with subsequent processing.

The present section of the paper provides the background needed for understanding the information which the algorithm integrating tense and aspect (presented in §4) is designed to compute. First, in §2.1, I explain what is meant by ***actual*** time and delimit the scope of the phenomena focused on here. Then in §2.2, I describe the components of temporal structure, and how they are used to distinguish states, transition events, and two ways of referring to processes. Also in this section I review Dowty's aspect calculus [Dowty79] and introduce how it is used in deriving the representation of temporal structure.

The remaining sections of the paper focus on the implementation. Section §3 describes the input to the temporal component. Section §4 presents the algorithm for computing the situation representations and their temporal location. Part of the computation of temporal location involves determining the Reference Time of a predication. Reference Time [Reichenbach47] pertains to the interpretation of relational temporal adverbials, i.e., adverbials which relate the time of a situation to another time (e.g. ***The ship was refuelled yesterday***, cf. [Smith81]).[7] Temporal connectives, for example, relate the time of a syntactically subordinate predication to a superordinate one. A brief discussion of how the Reference Time participates in the interpretation of temporal adverbial clauses introduced by connectives such as ***before, after*** and ***when*** is given in §5, whose more general topic is the utility of the situation representations for the interpretation of a variety of adverbial types.[8]

2.1. Actual Temporal Reference

Actual situations are those which are asserted to have already occurred, or to be occurring at the time when a text is produced. This excludes, e.g., situations mentioned in modal, intensional, negated or frequentative contexts.[9] A

---

[7] Reference time also plays a role in intersentential temporal reference (cf. Hinrichs, Moens and Steedman, Nakhimovsky, Webber, this volume)

[8] For the sake of brevity, the treatment of temporal adverbs with nominal complements is not described in this paper, but cf.[Dahl87b]

[9] These are not currently handled in the PUNDIT system. Predications embedded in any one of these contexts do not directly denote specific situations but rather denote types of situations which, e.g., might occur, have not occurred, or tend to occur. Treatment of these contexts awaits the development of a representation which distinguishes between

predication denotes an actual situation when two criteria are satisfied. First, at least one of the verb's arguments must be interpreted as specific [Dowty79, Mourelatos81, Vlach81]. For example, the simple past of ***fly*** denotes a specific situation in (1) but not in (2), because the subject of the verb in (2) is a non-specific indefinite plural.

(1) John flew TWA to Boston.

(2) Tourists flew TWA to Boston.

This paper does not address the interaction of the nature of a verb's arguments with the specificity of references to situations.

The second criterion is that the situation must be asserted to hold in the real world for some specific time. Predications in modal contexts (including the future; cf. 3) are excluded because their truth evaluation does not involve specific real-world times, but rather, hypothetical or potential times.

(3) The oil pressure should/may/will decrease.

Additionally, frequency adverbials like ***always*** may force a temporally non-specific reading, as in (4).

(4) John always flew his own plane to Boston.

PUNDIT's time component does not currently identify modal contexts, frequency adverbials, or non-specific verb arguments. However, it does identify predications denoting situation types when the form of the verb itself provides this information.

In evaluating actual time, PUNDIT distinguishes between examples like (5) and (6) on the basis of the verb and its grammatical categories. An actual use of the sentence in (5), for example, would report that a particular pump participated in a particular event at a specific time.

(5) The lube oil pump seized.

(6) The lube oil pump seizes.

(7) The lube oil pump seized whenever the engine jacked over.

Sentences (6) and (7), on the other hand, report on types of recurrent events. In example (7), it is the adverb ***whenever*** which indicates that the main clause refers to a recurrent type of event rather than to a specific event token situated at a particular time. In example (6), it is the lexical aspect of the verb ***seize*** in combination with the present tense which provides that information. A further difference between the two examples is that (7) entails that on at least one past occasion the pump actually seized when the engine jacked over, while (6) does not entail that the lube oil pump ever actually seized. We will see in §4.1 that (6) would immediately be determined not to evoke actual time on the basis of the lexical aspect of the verb and its inflectional form. Although PUNDIT does not yet handle frequency adverbials, §5 illustrates the procedure by which the main clause of (7) would be processed so that its relation to the subordinate

---

specific situations which hold for some real time and types of situations which hold for some potential time. One such proposal appears in [Roberts85], which allows for the creation of temporary contexts

clause event could be identified later.

Lexical aspect is the inherent semantic content of a lexical item pertaining to the temporal structure of the situation it refers to, and thus plays a major role in computing temporal information. The aspectual categories and their relevance to temporal processing are discussed in §2.2.

It should be noted that other semantic and pragmatic properties also affect temporal analysis. For example, there are conditions under which the present tense of a verb referring to an event, as in (6), is associated with an actual situation. Under the right conditions, first person performatives (e.g., ***I warn you not to cross me***) accomplish the named event at the moment they are uttered [Austin77]. Even a sentence like (6) can refer to an actual event if interpreted as a report of a presently unfolding situation, as in a sportscast. Handling tense in these types of discourse would require representing pragmatic features, such as the speaker/addressee relationship, in order to handle the relation of indexicals like tense and person to the speech situation [Jakobson57]. Section §3 briefly mentions some semantic distinctions pertaining to the verb in addition to lexical aspect which PUNDIT does handle. Otherwise, however, this paper focuses on temporal analysis of third person descriptions containing verbs whose arguments refer to specific, concrete participants.

2.2. Temporal Structure of Actual Situations

Situations are classified on the basis of their temporal structure into three types: states, processes and transition events. Each situation type has a distinct temporal structure comprised of one or more intervals. Two features are associated with each interval: kinesis and boundedness. Both terms will be defined more fully below, but briefly, kinesis pertains to the internal structure of an interval, or in informal terms, whether something is happening within the interval. Boundedness pertains to the way in which an interval is located in time with respect to other intervals, e.g., whether it is bounded by another interval.

This approach to the compositional semantics of temporal reference is similar in spirit to interval semantics in the attempt to account for the semantic effects of aspectual class[Dowty86, Dowty82, Dowty79, Taylor77]. However, interval semantics captures the distinct temporal properties of situations by specifying a truth conditional relation between a full sentence and a unique interval. The goal of PUNDIT's temporal analysis is not simply to sort references to situations into states, processes and events, but more specifically to represent the differences between the three situation types ***by considering in detail the characteristics of the set of temporal intervals that they hold or occur over*** [Allen84] (p. 132). Thus, instead of specifying a single set of entailments for each of the three situation types, the temporal semantics outlined here specifies what property of an interval is entailed by what portion of the input sentence, and then compositionally constructs a detailed representation of

a state, process or event from the intervals and their associated features. The critical difference from interval semantics is that while intervals are the fundamental unit from which situation representations are constructed, it is proposed here that intervals have properties which differentiate them from one another.

2.2.1. Situation Types and Temporal Structure

The three situation types—states, processes and transition events—are distinguished from one another entirely on the basis of the grammatically encoded means provided by the language for talking about how and when they occur. People certainly can and do conceptualize finer differences among real-world situations, and can even describe these differences, given sufficient time or space. But certain gross distinctions are unavoidably made whenever people mention things happening in the world. Here and in the next section we will examine the temporal distinctions encoded in the form of the verb, often referred to as ***aspect***, which are here referred to as temporal structure. Part of the temporal structure, that which Talmy described as the ***pattern of distribution of action through time*** [Talmy85], is represented in the time arguments for the three situation types. Another part of the temporal structure, its Event Time, is the component of temporal structure which gets located in time by tense and the perfect. All the relevant distinctions of temporal structure are represented in terms of intervals and moments of time.

States. Very briefly, a state is a situation which holds over some interval of time which is both stative and unbounded. A stative interval is one in which, with respect to the relevant predication, there is no change across the interval for which the situation holds. Thus, stative intervals are defined here much as stative predications are defined in interval semantics:

An interval I over which some predication ψ holds is stative iff it follows from the truth of ψ over I that ψ is true at all subintervals of I [Dowty86] (cf. p. 42).

Sentence (8) is an example of a typical stative predication whose verb phrase is headed by an adjective. During the interval for which the predicate ***low*** holds over the entity ***pressure***, each subinterval is equivalent to any other subinterval with respect to the asserted situation; thus, its kinesis is stative.

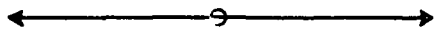
(8) The pressure is low.

Some of the diagnostic tests for stative predications are that they cannot be modifed by rate adverbials (****The pressure was quickly low***), nor referenced with ***do it*** anaphora (***The pressure was very low.*** ****The temperature also did it/that.***). While inability to occur with the progressive suffix has often been cited as another diagnostic, it is a less reliable one. Dowty [Dowty79] identifies a class of locative stative predications which occur in the progressive (e.g., ***The socks are lying under the bed***). Predicates denoting cognition or behavior have often been classified as statives but may occur in the progressive with

reference to a cognitive or behavioral process.[10] Although such verbs do not appear in the current domain, they would be treated differently from pure stative verbs.

The intervals associated with states are also inherently unbounded, although a temporal bound could be provided by an appropriate temporal adverbial (e.g., ***The pressure was normal until the pump seized***).[11] When an unbounded interval is located with respect to another point in time, it is assumed to extend indefinitely in both directions around that time, as with the punctual adverbial in (9). The moment within the interval which is explicitly located by tense and the punctual adverbial is the situation's Event Time, depicted as a circle in the middle of the interval, with arrows representing that the interval extends indefinitely into the past and towards the present.

(9) The pressure was low at 0800.



Situation type: state
Kinesis: stative
Boundedness: unbounded

This sentence would be true if the pressure were low for only an instant coincident with ***0800***, but it is not asserted to hold only for that instant; one thus assumes that it was low not only at the named time, but also prior and subsequent to it. In this sense, the interval is unbounded, as represented graphically above.

Processes. A process is a situation which holds over an active interval of time. Active intervals contrast with stative intervals in that there is change within the interval, a useful distinction for interpreting manner adverbials indicating rate of change, e.g., ***slowly*** and ***rapidly***. Since states denote the absence of change over time, they cannot be modified by rate adverbials; processes can be.

The definition of active intervals is also adapted from the characterization of process predications in interval semantics:

An interval I over which some predication ψ holds is active iff it follows from the truth of ψ at I that ψ is true over all subintervals of I down to a certain limit in size [Dowty86] (cf. p. 42).

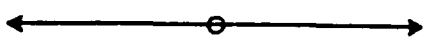
Active intervals can be unbounded or unspecified for boundedness, depending on whether the verb is progressive. In (10), the active interval associated with the alarm sounding is unbounded, and bears the same relationship to the named clock time as does the stative interval in (9) above.

---

[10] Cf. discussion of examples like *I am thinking good thoughts*, and *My daughter is being very naughty*, in [Smith86].

[11] In general, temporal adverbials can modify an existing component of temporal structure or add components of temporal structure.

(10) The alarm was sounding at 0800.

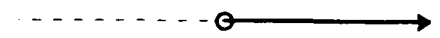


Situation type: process
Kinesis: active
Boundedness: unbounded

Progressive aspect has often been compared to lexical stativity.[12] Here the commonality among sentences like (9) and (10) is captured by associating the feature of unboundedness both with stative lexical items and with progressive aspect. The temporal structures of states and unbounded processes are thus identical with respect to boundedness. However, the distinction between the kinesis of (9) and (10) is retained by distinguishing active from stative intervals.

In (11) the interval associated with the alarm sounding is unspecified for boundednesss, meaning that the clock time may occur within the interval for which the alarm sounded, or at its onset or termination.

(11) The alarm sounded at 0800.



Situation type: process
Kinesis: active
Boundedness: unspecified

In (10), where the verb is progressive, the clock time is interpreted as falling within the unbounded interval of ***sounding*** but in (11), where the verb is not progressive, the clock time can be interpreted as falling at the inception of the process or as roughly locating the entire process.[13] Non-progressive forms of process verbs exhibit a wide variation in the interpretation of what part of the temporal structure is located by tense. The influencing factors seem to be pragmatic in nature, rather than semantic. The solution taken here is to characterize the Event Time of such predications as having an unspecified relation to the active interval associated with the denoted process, represented graphically above by the dashed line around the Event Time.

Transition Events. A transition event is a complex situation consisting of a process which culminates in a new state or process. The new state or process comes into being as a result of the initial process. Since states have no kinesis, they cannot culminate in new situations. The temporal structure of a transition event is thus an active interval followed by—and bounded by—a new

[12]For comparisons of stativity and the progressive, cf. [Vlach81] where the two are equated, Smith's counterargument [Smith86], and the interesting proposal in [Mufwene84].

[13]Nakhimovsky (this volume) makes essentially the same argument, namely that English lacks overt perfective grammatical aspect. In other words, the indeterminacy associated with the simple past of a process verb is evidence for the argument that the ***perfective*** or ***culminated*** reading associated with simple past transition event verbs, which are discussed in the next subsection, is a consequence of the interaction between tense and aspect, rather than of the simple past tense itself

active or stative interval.[14]

That there are these three distinct components of transition events can be illustrated by the following sentences in which the time adverbials modify one of the three temporally distinct parts of the predicated event.

(12) It took 5 minutes for the pump to seize.

(13) The pump seized precisely at 14:04:01.

(14) The pump was seized for 2 hours.[15]

The duration ***5 minutes*** in (12) above applies to the interval of time during which the pump was in the process of ***seizing***. The clock time in (13) corresponds to the moment when the pump is said to have made a transition to the new state of ***being seized***. Finally, the measure phrase in (14) corresponds to the interval associated with the new state.

Following [Dowty86], Vendler's two classes of achievements and accomplishments [Vendler67] are collapsed here into the single class of transition events, and for much the same reasons. That is, achievements differ from accomplishments in being ***typically of shorter duration*** and in not entailing a sequence of sub-events, but they nevertheless ***do in fact have some duration*** [Dowty86] (p. 43). Even so-called punctual events (e.g., ***They arrived at the station***, ***She recognized her long-lost friend***) can be talked about as if they had duration [Talmy85, Jackendoff87], apparently depending on the granularity of time involved. It is my belief that handling granularity depends on appropriate interaction with a relatively rich model of the world and of the current discourse, but would not require new units of time; depending on the level of detail required, moments could be exploded into intervals, or intervals collapsed into moments. For these reasons, punctual events are not treated here as a separate class.

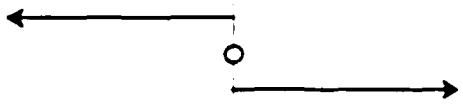
With verb's in Vendler's class of achievements, the same participant generally participates in both the initial process and the resulting situation, as in (15):

---

[14] This treatment of transition events closely resembles the ***event structure*** which Moens and Steedman refer to as a ***nucleus***. They define a nucleus as a structure comprising a culmination, an associated preparatory process, and a consequent state [Moens87].

[15] The durational adverbial in (14) forces a stative reading for a predicate which in isolation would be ambiguous between a passive and the adjectival passive [Levin86], in which the past participle is interpreted statively or adjectivally

(15) The engine failed at 0800.



Situation type: transition event
Kinesis: active
Boundedness: bounded

Here, the engine participates in some process (***failing***) which culminates in a new state (e.g., ***being inoperative***). In each case, however, there are two temporally distinct intervals, as shown in the diagram above, one bounded by the other.

Causative verbs typically denote accomplishments involving sub-events in which the action of one participant results in a change in another participant, as in (16):

(16) The pump sheared the drive shaft.

Here, a process in which the pump participated (***shearing***) is asserted to have caused a change in the drive shaft (***being sheared***). The consequence of the different argument structures of (15) and (16) on the event representation is discussed in the next section.

The boundary between the two intervals associated with a transition event, the transition bound, is defined as a transitional moment between the initial active interval and the ensuing active or stative interval associated with the new situation. An important role played by the transition bound is that it is the temporal component of transition events which locates them with respect to other times. For example, (15) asserts that the moment of transition to the new situation coincides with ***0800***. In contrast with examples (9)-(11), the status of the engine prior to ***0800*** is asserted to be different from its status at ***0800*** and afterwards. The components of temporal structure proposed here are intended to provide a basis for deriving what is said about the relative ordering of situations and their durations, rather than to correspond to physical reality. Thus a transition bound is a convenient abstraction for representing how transition events are perceived and talked about. Since a transition event is one which results in a new situation, there is in theory a point in time before which the new situation does not exist, and subsequent to which the new situation does exist. This point, however, is a theoretical construct not intended to correspond to an empirically determined time. It corresponds exactly to the kind of boundary between intervals involved in Allen's ***meets*** relation [Allen83, Allen84].

2.2.2. Dowty's Aspect Calculus

The intervals for which situations hold are closely linked with the semantic decompositions of the lexical items used in referring to them.

This allows PUNDIT to represent precisely what kinds of situations entities participate in and when. The decompositions include not only *N*-ary relational predicates among the verb's arguments [Passonneau86], but also the aspectual operators for processes and events proposed by Dowty [Dowty79]. The main clauses for examples (9), (10), (15) and (16) are given below as examples (17)-(20).

(17) The pressure was low.
Decomposition: low(patient([pressure1]))

(18) The alarm was sounding.
Decomposition: do(sound(actor([alarm1])))

(19) The engine failed.
Decomposition: become(inoperative(patient([engine1]))

(20) The pump sheared the drive shaft.
Decomposition: cause(agent([pump1]),become(sheared(patient([shaft1]))))

In (17), the semantic predicate **low** is associated with the predication ***be low***, and is predicated over the entity referred to by the subject noun phrase, ***the pressure***.[18] The time component recognizes this structure as a stative predication because it contains no aspectual operators.

The decomposition for (18) consists of a basic semantic predicate, **sound**, its single argument, and the aspectual operator ***do*** indicating that its argument is in the class of process predicates; the actor role designates the active participant.

The decompositions of transition event verbs contain the aspectual operator ***become*** whose argument is a predicate indicating the type of situation resulting from the event. With inceptive verbs, as in (19), the actor of the initial process is also the patient or theme of the resulting situation, although this dual role is not represented explicitly in the decomposition. If a distinct actor causes the new situation, the verb falls into the class of causatives and the actor of the initial process is conventionally called an agent, as in (20). Other decompositional analyses [Dowty79, Foley84] conventionally represent the initial process of transition event verbs by associating an activity predicate (e.g., ***do***) with the actor or agent of the initial process (e.g., ***cause(do(agent( )), become(inoperative(patient( ))))***). The decompositions in (19) and (20) can be considered abbreviated versions of these more explicit predicate/argument structures.

The ***become*** operator of transition event verbs thus provides a crucial piece of information used when deriving representations of transition events. Given a reference to a specific transition event which has already taken place, the temporal component deduces the existence of the new situation that has come into being by looking at the predicate embedded beneath the ***become*** operator. This is described more fully in §4.2.3.

---

[18] The atom ***/pressure1/*** is a pointer for the entity referred to in the noun phrase and is created by PUNDIT's

As will be shown in §4, PUNDIT represents actual situations as predicates identifying the situation type as a state, process or event. In order to familiarize the reader with the representation schema without needless repetition of detail, a single example of a situation representation is given below for (17).

(17) The pressure was low.

```
state([low1],
      low(patient([pressure1])),
      period([low1]))
```

Each situation representation has three arguments: a unique identifier of the situation, its semantic decomposition, and its time argument, in this case, the interval (or ***period***) over which the predicate holds. The same pointer (e.g., [low1]) is used to identify both a specific situation and its time argument because the actual time for which a situation holds is what uniquely identifies it. The participants in a situation help distinguish it from other similar situations, but while the same entities can participate in other situations, time never recurs.

Having introduced the distinct situation types and the temporal structures which distinguish them, the next steps are to show how they are computed, and how they permit a simple computation of temporal location. This will be done in §4. Since the preceding discussions also introduced the representation of lexical aspect and the relevance of the verbal categories, it is now possible to clearly summarize the input which the temporal analysis component receives.

3. Input to the Temporal Component

PUNDIT's time component performs its analysis after the sentence has been parsed and recursively after the semantic decomposition of each predicating element in the sentence has been created [Palmer86]. Although this paper focuses on the temporal analysis of certain kinds of tensed verbs, the basic algorithm described here has been extended to handle other cases as well. Describing the full input to the temporal component provides an opportunity to mention some of them.

The input to the time component for each tensed clause includes not only the surface verb and its tense and aspect markings, but also the decomposition produced by analyzing the verb and its arguments (cf. §2.2.2.). The input to the time component is thus a list of the following form:

[Tense, Perfect, Progressive],Verb,Decomposition,{Context}

Each element of the list will be described in turn.

---

reference resolution component [Dahl86]

3.1. Verbal Categories

The first element in the input list is itself a list indicating the form of the verb, i.e., its grammatical inflection.

[[**Tense, Perfect, Progressive**],Verb,Decomposition,{Context}]

The tense parameter is either past or present.[17] If the verb is in the progressive or perfect, the corresonding parameter appears while absence of either in the input sentence is reflected in its absence from the list.

3.2. Three Orders of Verbs

The next two elements in the input to the time component are the surface verb and its decomposition. Lexical aspect is encoded in the decomposition as described in §2.2.2 for the cases where it is relevant. However, it is a more fundamental classification pertaining to the verb which helps determine the cases where aspect is relevant.

[[Tense, Perfect, Progressive],**Verb**,Decomposition,{Context}]

Since this information is only for treating more complex cases than are described in this paper, the following discussion is intended only to indicate that the model has been extended to cover verbs whose semantic structure contains temporal information of a different order than the inherent temporal structure of an actual situation. After a brief description of three temporal orders of verbs, the discussion will return to explication of the input required for implementing the basic model.

In addition to the aspectual distinction among state, process and transition event verbs, there are other distinctions related to temporal semantics. A particularly significant one is among what I call first, second and third order verbs, by analogy with the distinction among first, second and third order logics. A first order verb is one whose arguments are concrete entities, e.g., humans, machines, and other physical objects. A second order verb takes as its arguments states, processes and events, but does not in and of itself refer to a situation. Rather, its semantic content is primarily temporal or aspectual (e.g., ***occur, follow***). Third order verbs refer to complex situations (e.g, ***result, cause***) whose participants are themselves situations. The aspectual distinctions among verbs referring to states, processes and transition events are only relevant to first order verbs.

Second order verbs can be identified by the impossibility of temporal modification of a situation referred to by the verb, independent of the situation(s) referred to by the verb's argument(s) [Newmeyer75], as can be seen by contrasting examples (21) and (22) with (23) and (24).

(21) The failure occurred on Tuesday.

---

[17] For sentence fragments such as ***erosion of blade tip evident***, the tense parameter is actually ***unlensed***. The time component assigns present or past tense readings to fragments, depending on the aspectual class of the fragment [Linebarger88].

(22) The failure was discovered on Tuesday.
(23) *The failure that happened on Monday occurred on Tuesday.
(24) The failure that happened on Monday was discovered on Tuesday.

Example (22) mentions two distinct situations, a ***discovery*** and a ***failure***. In (21), however, the subject of the sentence, ***the failure***, denotes an event, but the verb ***occur*** does not denote a separate situation. It provides tense and aspect information for interpreting its argument. In other words, the temporal information in (21) is very similar to that contained in (25):
(25) Something failed on Tuesday.

A pragmatic difference between the two sentences is that in (21) it is not necessary to mention what failed whereas in (25), the verb ***fail*** must have a subject. Other verbs in this class are ***follow***, ***precede***, ***continue***, ***happen*** and so on. Because these verbs contribute primarily temporal information, they are conventionally referred to as aspectual verbs [Freed79, Lamiroy87, Newmeyer75].

It is easy to see that the analysis of aspectual verbs must be implemented somewhat differently from verbs like ***fail***, which directly denote situations. In a sentence like (25), the relevant temporal information is contained in the verb and its tense and aspect marking alone. In contrast, the temporal information in (24) pertaining to the ***fail*** event is distributed not only in the verb and its tense and aspect markers, but also in its subject. Temporal analysis of sentences like (24) must be performed not only at the main clause level, but also at the level of embedded propositions. In essence, analysis of aspectual verbs is of a different order. Consequently, verbs like ***fail*** are classified here as first-order verbs while the so-called aspectual verbs are classified as second-order.

PUNDIT's temporal component also handles a third class of verbs, classified as third-order. A third-order verb denotes a real-world situation, but its arguments are other situations. Consequently, the verb may contribute temporal information about the arguments as well as about the situation it denotes. The verb ***result*** illustrates this type. Sentence (26) asserts the existence of a **result** situation; the **result** relationship holds between an instigating situation mentioned in the noun phrase ***loss of air pressure***, and a resulting situation mentioned in the noun phrase ***failure***.
(26) Loss of air pressure resulted in failure.

Additionally, the meaning of ***result*** includes the temporal information that the instigating situation (the ***loss***) precedes the resulting situation (the ***failure***). A full temporal analysis of sentences like (26) requires two steps. The first is to analyze the temporal structure of the situation denoted by the verb. The second is to draw the correct temporal inferences about the verb's propositional arguments. Such verbs combine some of the properties of both first and second-order verbs and thus constitute a third order of analysis. Classifying a verb as a third-order verb drives the search for temporal inferences associated with its arguments.

The classification of these three orders of verbs, summarized in Table 1, is recorded independently of the lexical decompositions used by both the temporal analysis component and the semantic interpreter. At present, verb order information is used only by the temporal analysis component. It essentially selects for the appropriate flow of control through the temporal processing procedures. Although PUNDIT recognizes the distinction between first, second and third order verbs and processes the relevant temporal information in each case, the remainder of the paper will deal only with the analysis of first-order verbs.

3.3. Lexical Aspect

The third element in the input list is the decomposition structure produced by the semantic analysis of the verb and its arguments.

[[Tense, Perfect, Progressive], Verb, **Decomposition**, {Context}]

The important aspectual features of the decompositions, discussed in §2.2.2, can be summarized as follows. If the decomposition of a first-order verb contains a ***become*** operator, the verb is in the transition event class; otherwise, if it contains a ***do*** operator, the verb is in the process class; else, the verb (or other predicate) is stative.

3.4. Discourse Context

The final element in the input to the temporal component is a data structure representing the current discourse context.

[[Tense, Perfect, Progressive], Verb, Decomposition, **{Context}**]

The first element of this data structure is a list of unanalyzed syntactic constituents. At this stage of processing, PUNDIT has produced a full syntactic analysis of a surface sentence (or sentence fragment), and a semantic decomposition of some predication within the sentence. After the semantic analysis of a clause, the constituent list contains all those syntactic constituents which do

Table 1
Three orders of verbs

Order	Examples	Definition
First	'fail', 'operate'	verbs which denote situations and whose arguments are not propositional
Second	'occur', 'follow'	verbs which provide temporal information about their propositional arguments
Third	'result', 'cause'	verbs which denote situations but which also provide temporal information about their propositional arguments

not serve as arguments of the verb, e.g., adverbial modifiers of the verb phrase and sentence adjuncts. After the analysis of the main clause of sentence (27) for example, the constituent list would contain two unanalyzed constituents, the prepositional phrase introduced by ***during***, and the subordinate clause introduced by ***when***.

(27) The pump failed during engine start, when oil pressure dropped below 60 psig.

This list of constituents is processed after the temporal content of a predication is analyzed in the search for temporal adverbials which modify the predication (cf. §5 below). The data structure representing the current discourse context contains temporally relevant information, such as the tense and voice of the main clause. The main clause tense is used for the analysis of situations mentioned in embedded tenseless constituents while voice is used in analyzing adjectival passives.

The next section describes an algorithm for interpreting the four pieces of information relevant to actual references to states, processes and events. It demonstrates how the temporal structure and temporal location are generated from the verb's grammatical categories of tense, perfect and progressive, and from its lexical aspect.

4. Algorithm for the Temporal Analysis of Inflected Verbs

The introductory and discussion sections have undoubtedly reinforced the view that semantic processing of temporal information is a complicated problem, even when the scope of the problem is constrained to the simple cases addressed here. Relevant information is distributed within and across distinct constituents, and their contribution to temporal information can depend upon co-occurring elements. Yet these are in no way insurmountable problems. The fundamental design principles behind my approach to temporal processing have been to carefully separate the analysis into distinct subtasks, to pare down to a minimum the information available to each task, and to provide a simple compositional semantics for each kind of temporal input. In this section, I outline the basic algorithm for the temporal analysis of inflected verbs. This algorithm analyzes the four components of the inflected verb described in the preceding section (lexical aspect, progressive, perfect, tense). The output which is generated can then serve as input for further temporal processing. Section §5 illustrates the integration of this basic algorithm into a more global procedure that successively interprets the main and subordinate clauses of complex sentences where the subordinating conjunction is a temporal adverbial.

The basic algorithm for the temporal analysis of inflected verbs has a simple tri-partite control structure designed to answer three distinct questions:

1) Does the predication denote a specific situation with ACTUAL TIME reference?

2) If so, what is the TEMPORAL STRUCTURE of the situation, i.e., how does it evolve through time and how does it get situated in time?

3) Finally, what is the TEMPORAL LOCATION of the situation with respect to the time of text production, and what is the temporal vantage point from which the situation is described?

Figure 1 illustrates the algorithm's global control structure, with the modules corresponding to each question as well as the relevant input for each module. The first module examines all four temporal parameters described in sections §§3.1 and 3.3 in order to reject certain cases. The second module requires only the two parameters pertaining to the computation of temporal structure. It sends a component of the temporal structure, the Event Time, to the third module which locates the Event Time by analyzing the remaining two temporal parameters, Tense and Perfect.

4.1. Module 1: Actual Time

The first task performed by PUNDIT's temporal component is to identify references to specific situation tokens, that is, instances of situations which have actually occurred. The input is the lexical verb and its grammatical categories. In certain cases, the form of the verb itself can indicate that the predication refers to a type of situation, rather than to a specific token. Thus the screening step described here rejects these cases, and otherwise assumes that the predication denotes a specific situation. As pointed out in §2.1, the verb itself provides insufficient information in two kinds of cases: those where explicit disconfirming information occurs elsewhere in the sentence (e.g., arguments of the verb, modals, frequency adverbials; cf. examples (2) and (7), repeated below),

(2) Tourists flew TWA to Boston.

(7) The lube oil pump seized whenever the engine jacked over.

and those where pragmatic features of the discourse context affect the interpretation of semantic input (as in a sportscast). While Module 1 currently serves only as a filter, it could be made to generate informative output for subsequent processing of semantically and pragmatically more complex phenomena.

In Section §2.1 it was shown that two classes of inflected verbs generally denote situation types, rather than actual tokens. These are process verbs and transition event verbs in the simple present tense (i.e., non-progressive and non-perfect), as exemplified in (28) and (29).

(28) Number 2 air compressor operates at reduced capacity.
[***operate*** is a process verb]

(29) They replace the air compressor every three years.
[***replace*** is a transition event verb]

For the compound tenses, present tense interacts with the progressive and perfect verbal categories. The progressive alters the aspectual properties of non-stative verbs so that they refer to unbounded situations, and unbounded

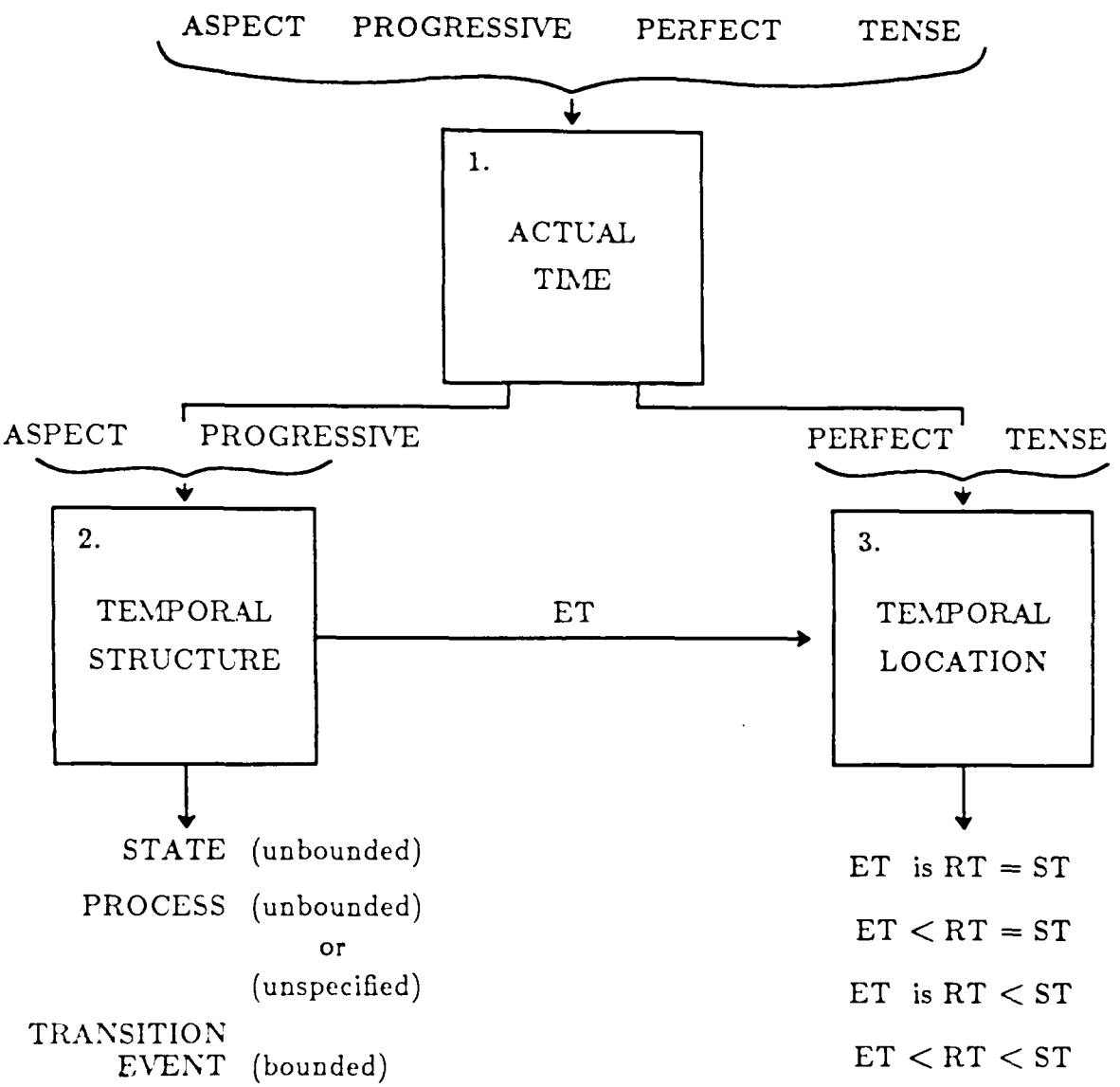


Figure 1. Algorithm for temporal analysis of inflected verb

situations—unlike the other temporal structures—can be located in the actual present (cf. §4.2.2). With the perfect forms, the situation being referred to is always located in the past, and tense pertains to the situation's Reference Time rather than its Event Time (cf. 2.3). Thus, as shown in Fig. 1, all four elements in the temporal data structure are inspected in order to identify the two cases exemplified in (28) and (29).

Table 2 summarizes the relation between the inflected verb and actual temporal reference.

TABLE 2
MODULE 1. ACTUAL TIME

LEXICAL ASPECT	PROGRESSIVE	PERFECT	TENSE	ACTION
Non-stative	no	no	present	reject
				accept

In the current implementation of PUNDIT, predications which meet the first condition do not receive further temporal analysis.

4.2. Module 2: Compute Temporal Structure

Module 2 computes the first type of specific temporal information associated with reference to an actual situation. It generates an explicit representation of the situation's temporal structure. This structure includes one or more time arguments associated with the semantic predicates in the decomposition, and the situation's Event Time. Each situation type—state, process, transition event—receives an appropriate situation label, time argument(s), and Event Time. The temporal structure evoked by an inflected verb can be computed entirely on the basis of the values of the two aspectual elements in its input (Lexical Aspect, Progressive), as shown in Fig. 1. The algorithm for Module 2, summarized in Table 3, will be described in the following three sections corresponding to the three situation types.

TABLE 2 2. TEMPORAL STRUCTURE				
LEXICAL ASPECT	PROGRESSIVE	LABEL	TIME ARGUMENT	EVENT TIME
stative	Yes/No	State	unbounded stative interval	includes ET
process or transition event	Yes	Process	unbounded active interval	includes ET
process	No	Process	unspecified active interval	has ET
transition event	No	Event	transition bound	*unifies with* ET

Though not shown in the figure or in Table 3, Module 2 also receives another input data structure: the semantic decomposition. The decomposition is analyzed during the processing of transition event situations in order to associate distinct time arguments with distinct semantic predicates in the decomposition. This procedure is explained in the appropriate section below.

4.2.1. States

As shown in Table 3, if the lexical aspect of the predicate is stative (Aspect=stative), then the progressive parameter is irrelevant for computing temporal structure. Lexical stativity is sufficent to identify the situation as a state whose time argument is an unbounded stative interval.

Example 30 gives a simple stative sentence, the relevant input to Module 2, and the final situation representation. Note that (30) illustrates the use of the progressive with a verb in the locative class of statives noted by Dowty [Dowty79], and mentioned in §2.2.1.[18]

(30) Metallic particles are clogging the strainer.

Lexical Aspect = stative

Situation Representation:

```
state([clog1],
    clog(instrument([material1]), theme([strainer2])),
    period([clog1]))
```

As soon as the lexical aspect is recognized to be stative, Module 2 generates the state label and period time argument used in creating the representation depicted above. A period time argument in the context of a state representation denotes a stative interval. The situation representation in (30) indicates that a specific state, ***clog1***, holds over the stative interval, ***period([clog1])***; the decomposition in the representation indicates the participants and the relation between them that holds over this interval. By definition, this interval also has an Event Time associated with it, whose relation to the interval we can

[18] As noted elsewhere, the aspectual classification of verbs is not completely determinate [Talmy85]. *Clog* may very well be a verb that can refer either to a process or a state, and it might be possible to decide dynamically the lexical aspect of a specific instance through interaction with a sophisticated model, such as one which incorporates the notion of resource use, as suggested by Nakhimovsky (this volume). However, in PUNDIT the aspectual classification of verbs is domain dependent.

determine by its boundedness feature.

Stative intervals are assumed to be unbounded unless an endpoint is provided by further processing (e.g., through adverbial modification, inference). For unbounded intervals, the Event Time is always an arbitrary moment ***included*** within the interval. This is represented as a binary predicate of the following form, where the ***moment*** time argument is the Event Time:

Event Time = moment([clog1])
such that includes(period([clog1]), moment([clog1]))

This predicate and the state representation given above exemplify the output of Module 2 for state situations. The Event Time generated here is then passed to Module 3 in order to determine its temporal location. We will return to this same example in the discussion of temporal location (§4.3).

4.2.2. Processes

There are three surface forms which denote process situations: non-progressive process verbs, progressive process verbs, and progressive transition event verbs. The non-progressive and progressive cases have distinct temporal structures, due to differences in the relation of the Event Time to the active interval over which the process holds. Since this is the only difference among the three cases, the similarities in temporal structure will be presented before the Event Time is discussed.

A non-stative predication which either has a process verb or is in the progressive (i.e., the three combinations of non-progressive process, progressive process, and progressive transition-event) evokes a process representation. Thus, the following three example sentences would each be represented with a process label and a period time argument, representing the active interval over which the process holds. Examples (31) and (32) illustrate the two forms of process verbs that evoke process situations; since they receive the same representation, it is shown only once. Example (33) shows the third type of reference to a process, with a progressive transition event verb.

(31) The diesel operated.
Lexical Aspect = process
Progressive = no

(32) The diesel was operating.
Lexical Aspect = process
Progressive = yes

(31-32) Situation Representation:

process([operate1],
do(operate(actor([diesel1]))),
period([operate1]))

(33) The pump is failing.
Lexical Aspect = transition event
Progressive = yes
Situation Representation:

process([fail1],
become(inoperative(patient([fail1]))),
period([fail1]))

The process representation for (33) contains the full decomposition for the verb ***fail*** with its aspectual operator ***become***. In this context, the ***become*** operator does not denote a transition to a new situation, but rather, indicates a process of ***becoming*** which might or might not culminate in such a transition.

Referring again to Table 3, we note that the active intervals for both (32) and (33) will be unbounded, in contrast to (31), where the active interval is unspecified for boundedness. The consequence of this difference on the representation of the Event Time is outlined in the following paragraphs.

Unbounded processes. The predicate specifying the relation between the Event Time of an unbounded process, and the period over which the process holds, is identical to that for states. That is, the period time argument ***includes*** an arbitrary moment which serves as the situation's Event Time, as shown below.

(32) The diesel was operating.
Event Time = moment([operate1])
such that includes(period([operate1]), moment([operate1]))

(33) The pump is failing.
Event Time = moment([fail1])
such that includes(period([fail1]), moment([fail1]))

The progressive always implies unboundedness, and in this respect resembles lexical statives. Again, it is important to remember that an unbounded interval can acquire endpoints through further processing (e.g., of temporal adverbials, as in ***The diesel was operating until the pump failed***).

Unspecified processes. For non-progressive process verbs, the period associated with the predication is unspecified for boundedness (cf. discussion of example (11) in §2.2.1). This gives rise to an indeterminate relationship between the Event Time and the period time argument over which the process holds; i.e., the Event Time may start, end, or be included within the period. This unspecified relationship is represented by means of a binary **has** predicate, as shown for example (31).

(31) The diesel operated.
Event Time = moment([operate1])
such that has(period([operate1]), moment([operate1]))

Both Event Time predicates given so far (i.e., ***includes***, ***has***) indicate a relation between an arbitrary moment and a single interval over which a state or process holds. There is otherwise nothing distinctive about the moment selected to be the Event Time of a process or state situation. In contrast, as Table 3 indicates, and as discussed in §2.2.1, the Event Time of a transition event is equated with a distinctive component of its temporal structure, viz., the transition bound between a process that initiates the event and the new situation reached at the culmination of the process.

4.2.3. Transition Events

Table 3 shows only one component of the temporal structure of a transition event (the relevant line of the table is repeated below).

LEXICAL ASPECT	PROGRESSIVE	LABEL	TIME ARGUMENT	EVENT TIME
transition event	No	Event	transition bound	***unifies with*** ET

As noted in §2.2.1, a transition event has three temporal components: an initial active interval leading up to a transition, the moment of transition, and the interval associated with the new, resulting situation. In theory, then, one could represent the full temporal structure of a transition event predication (e.g., ***The pump failed***) as three contiguous states of affairs: an initial process (e.g., ***failing***) leading up to a transitional moment (e.g., ***becoming inoperative***) followed by a new state of affairs (e.g., ***inoperative***). At present, PUNDIT explicitly represents only the latter two components of transition event predications: the moment (transition bound) associated with an event of becoming, and the period associated with the resulting situation. This representation has been found to be adequate for the current applications. Thus transition events are actually assigned two situation representations: an event representation with a moment time argument, represented with the input decomposition, and a resulting state or process situation with a period time argument, for which a new decomposition is derived from the input decomposition. Example (34) illustrates a typical transition event sentence, the relevant input for computing temporal structure, and the two situation representations.

(34) The pump failed.
Lexical Aspect = transition event
Progressive = no
Situation Representation:

```
event([fail1],
    become(inoperative(patient([pump1]))),
    moment([fail1])
```

Situation Representation:

```
state([fail2],
    inoperative(patient([pump1])),
    period([fail2]))
```

The first situation representation corresponds to the transition event itself. Module 2 generates the event label and moment time argument used in creating the type of event representation shown above for non-progressive transition event verbs. The moment argument of a transition event is the transition bound implying the onset of a new situation. When Module 2 creates an event with a moment argument, it also creates a representation for the implied situation. In example (34), the new situation is a state. When creating the representation for the situation resulting from a transition event, it is necessary to determine the appropriate situation label, time argument, and semantic decomposition for the new situation. This is where the semantic decomposition for transition events plays a role, as will be described below.

All transition event verbs contain a state or process predicate embedded beneath an instance of the aspectual operator ***become***. The full decomposition represents the type of situation associated with the moment of transition. The portion embedded beneath ***become*** is the situation type associated with the new situation. For example, the decomposition passed to the time component for sentence (34) would be:

```
become(inoperative(patient([pump1]))).
```

As shown in (34), this decomposition appears in the representation of the transition event itself. The argument to the ***become*** operator is then extracted for use in the new situation representation:

```
inoperative(patient([pump1]))
```

The extracted decomposition is inspected to determine its aspectual class, completely analogously to the procedure for determining the aspectual class of the input predicate (cf. §3). In this case, the embedded predicate decomposition is stative because it contains no aspectual operators. If it contained the ***do***

operator, the new situation would have been a **process**.[19] In this fashion, the decomposition guides the selection of the situation label and time argument for the situation inferred to result from the transition event.

The final piece of temporal structure derived for a transition event is the temporal relation between the moment associated with the transition event (e.g., moment([fail1])) and the period associated with the resulting situation (e.g., period([fail2])). The event moment is the onset of the period. Following [Allen83], this is called a ***start*** relationship. By definition then, every transition bound starts some period. In the case of example (34), the moment of failure starts the period for which the pump is in an inoperative state.

start(moment([fail1]), period([fail2]))

The Event Time of a transitional event is always identified with the transition bound. Thus for examples like (34), the moment time argument serves as the Event Time of the transition event. This identity relation is not represented as a predicate, but is rather handled via unification, as indicated in Table 3.

4.3. Module 3: Compute Temporal Location

PUNDIT's temporal component employs a Reichenbachian analysis of tense whereby situations are located in time in terms of three temporal indices: the Event Time, Speech Time, and Reference Time.[20] It diverges from Reichenbach primarily by distinguishing between the Event Time and the temporal structure of a situation. While Reichenbach acknowledged that the progressive, for example, pertains to temporal duration [Reichenbach47], he did not discuss the differences in temporal structure associated with distinct situation types and their interaction with tense. Here, the Event Time is only a single component of the full temporal structure of a situation. In this section, we will see how this method of defining the Event Time makes it possible to compute temporal location independently of lexical or grammatical aspect while preserving the distinctive temporal information they contribute to references to actual situations.

The tense and perfect parameters specify the sequencing relations among the Event Time, Reference Time and Speech Time, with each of the four configurations of tense and perfect specifying a distinct ordering, as shown in Fig. 1 and repeated below:

---

[19] The embedded decomposition never contains the ***become*** operator, a decomposition with two ***become*** operators (e.g., ***become(become(inoperative(patient([pump1])))))***) would be incoherent

[20] Reichenbach's treatment [Reichenbach47] of tense and other ***token reflexive*** (indexical) elements is similar to Jakobson's [Jakobson57]

ET is RT = ST	simple present
ET < RT = ST	present perfect
ET is RT < ST	simple past
ET < RT < ST	past perfect

The Speech Time, or time of text production, is given. It serves as the temporal fulcrum with respect to which the other temporal indices are located. As shown in Table 4, the presence or absence of the perfect indicates whether the Event Time and Reference Time are distinct, in which case the Event Time precedes the Reference Time, or whether they are identical. Tense is taken to indicate the relation between the Reference Time and the Speech Time, following Reichenbach's suggestion: ***the position of R[T] relative to S[T] is indicated by the words "past", "present", and "future"*** [Reichenbach47].

TABLE 4
8. TEMPORAL LOCATION

PARAMETER	VALUE	RULES
Perfect	Yes	precedes(ET,RT)
	No	ET is RT
Tense	Past	precedes(RT,ST)
	Present	coincide(RT,ST)

Since we are dealing here with actual time, rather than potential or hypothetical time, there is only past or present. That is, the Reference Time either precedes or coincides with the Speech Time.

The Reference Time and the Event Time are identical to one another for the simple tenses (ET is RT), which has the effect that tense applies to the Event Time. Thus, for the simple present, the Event Time and the Speech Time ***coincide***. Note that a distinction is made here between ***identity*** and ***coincidence*** of distinct indices. For any speech act or text containing a description of a situation, the speech situation and the described situation are always conceptually and observationally distinct, thus also their respective temporal indices. These indices are therefore represented as distinct times which, in the present tense, happen to coincide. However, with the simple tenses, there is no reason to create a distinct Reference Time and a relation saying that it coincides with the Event Time. Rather, there are two different functions which in the case of the simple tenses are filled by the same temporal index. The function of the Reference Time is explained more fully below.

Webber (this volume) reviews and expands upon the role Reference Time plays in intersentential temporal reference. Reference Time also plays a role in

interpreting relational adverbials like ***now***, ***yesterday***, ***when***, and so on. Adverbs like ***now*** and ***yesterday*** relate the Reference Time of a predication to an implicit time, viz., the Speech Time. Relational adverbs like ***before***, ***after*** and ***when*** relate the time of the predication they modify to an explicitly mentioned time, i.e., the Reference Time associated with their syntactic complements. In the absence of the perfect, the Reference Time is identical with the Event Time, as in (35) and (36).

(35) The pressure is normal now.

(36) The pressure was low yesterday.

In the perfect tenses, the Reference Time and Event Time are distinct. The Event Time of both the present and past perfect predications in (37) and (38) is past, i.e., the moment of ***failure*** is in the past.

(37) The pump has now failed.[21]

(38) The pump had failed when the gear began to turn.

With the present perfect, it is the Reference Time that is present, as shown in (37) by the admissibility of the adverb ***now***, which also refers to the present. On one reading of (38), the Event Time, or moment of failure, precedes the Reference Time, i.e., the time specified by the ***when*** clause. The perfect tenses can also be used simply to affirm truth or falsehood,[22] thus (38) has another reading in which the perfect does not contribute a distinct Reference Time, but merely asserts that it is in fact the case that the pump failed when the gear began to turn.

4.3.1. Simple tenses

The distinct relations of Event Time to temporal structure corresponding to the three categories of boundedness—unbounded, unspecified, and bounded—correlate with distinctive behavior of the present tense. If the temporal structure associated with a predication is an unbounded interval, the simple present locates some time within the interval coincident with the Speech Time. Examples (35)-(38) illustrate the simple present in the context of the four types of predications which hold over unbounded intervals.

(35) The pressure is low.
  Lexical aspect: stative
  Progressive: no

(36) Metal particles are clogging the strainer.
  Lexical aspect: stative
  Progressive: yes

(37) The pump is operating
  Lexical aspect: process

---

[21] Dowty and others have pointed out that the situation mentioned in a present perfect often persists up to the Speech Time [Dowty82]. However, this is generally not the case with reference to unbounded processes (e.g., ***The pump has operated***), and seems to depend on a variety of pragmatic factors for the other situation types.

[22] Cf. McCawley's discussion of the ambiguities of the perfect [McCawley71], especially the ***assertorial*** perfect.

Progressive: yes

(38) The pump is failing.

Lexical aspect: transition event

Progressive: yes

In these examples, the predicate is asserted to hold for some interval of unknown duration which includes the Speech Time. Since this interval corresponds by definition to actual time, it cannot be known to continue beyond the Speech Time into the future. However, that it can extend indefinitely into the past is illustrated by (39), where the situations referred to in the first and second conjuncts are assumed to be the same.

(39) The pressure is low and has been low.

Predications involving process or transition event verbs in the simple present have already been eliminated by Module 1 on the assumption that sentences like (40) and (41) do not refer to actual time.

(40) The pump operates.

Lexical aspect: process

Progressive: no

(41) The pump fails.

Lexical aspect: transition event

Progressive: no

If a predication is not explicitly unbounded, i.e., if it has or may have an endpoint, then the present tense cannot be interpreted as locating the Event Time in the actual present. An Event Time located within an unbounded interval corresponds to persistence of the same situation, whereas an Event Time that may also be an endpoint corresponds to a transition. The way in which examples like (40) and (41) are interpreted can be explained by considering that we cannot announce changes in the world at the exact moment that we perceive them, although in the guise of reportage or sportscasting, we act as though we can.

In contrast to the simple present, the simple past can locate the Event Time of any temporal structure prior to the Speech Time. What is distinctive about the past tense in the context of the different temporal structures pertains to the temporal structure surrounding the Event Time. If the temporal structure is an unbounded interval, then the Event Time is some moment prior to the Speech Time within a persisting interval, and the same situation extends unchanged forward towards the present and back into the past. Example (42) illustrates the lack of contradiction in asserting the continuation up to the present of the past, unbounded situation mentioned in the first clause.

(42) The pump was failing and is still failing.

The temporal structure associated with the situation mentioned in the first clause of (43), in the simple past, is an unspecified interval. Here it is unclear whether the two conjuncts refer to the same situation. Since the Event Time of the first conjunct is represented non-committally, i.e., it may or may not be an

endpoint of the interval, both interpretations are provided for by the representations generated here.

(43) The pump operated and is still operating.

Finally, the simple past of a predication denoting a transition event definitely locates an endpoint. The Event Time of (44) is the transitional moment between an initial process of ***failing*** and a resulting state of ***being inoperative***.

(44) The pump failed and is still failing.

The first clause of (44) is represented by PUNDIT to assert the following temporal information: there was a moment of transition at which the pump failed, viz. its Event Time (***moment([fail1])***); this moment started a period in which the pump was inoperative (***start(moment([fail1], period([fail2])))***); and finally, it preceded the Speech Time (***precedes(moment([fail1]), Speech Time)***). The second clause cannot refer to the same transition event because a unique transition bound cannot both precede and coincide with the Speech Time, nor can it both be an endpoint of, and contained within, an interval. Rather, the second clause refers to a distinct situation, either a process which the speaker presumes will eventually result in a new failure, or an iteration of successive failure events. Of these two possibilities for the second clause, PUNDIT currently generates only the former.

4.3.2. Perfect Tenses

The perfect tenses have a more complex semantics and pragmatics than the simple tenses. The semantic interpretation given here accounts for the temporal interpretations assigned to the perfect tenses in which the Event Time and Reference Time are distinct from one another. There are uses of the perfect which do not have these temporal effects, as pointed out in [McCawley71], i.e., cases where the Event Time and Reference Time would not be distinct. Here we consider only the temporally relevant uses of the perfect, where each perfect tense specifies two temporal relations: in both cases, the Event Time precedes the Reference Time; and tense indicates whether the Reference Time coincides with or precedes the Speech Time.

The following examples illustrate the present and past perfect with a variety of temporal structures. The only difference between these examples and the simple present tenses examined in the preceding section is the relation between the Reference Time and Event Time. The relation between temporal structure, Event Time and Speech Time is the same as for the simple past.

(45) The engine has been operating. (unbounded process)

(46) The engine has operated. (unspecified process)

(47) The pump has failed. (transition event)

(48) The pressure had been low. (state)

(49) The pump had failed. (transition event)

5. Interpreting Temporal Adverbials

It is assumed that temporal adverbials can be analyzed in terms of the same components of temporal structure and temporal sequencing constraints that apply to situations. The situation representations developed here provide a foundation for interpreting three distinct types of adverbial modification corresponding to the three features represented in temporal structure, i.e., kinesis, intervals, and moments. Rate adverbs like ***slowly*** and ***rapidly***, which modify the manner in which situations evolve through time, modify active intervals and not stative intervals. For an example like (50), no explicit active interval would be represented, thus one would have to be coerced in order to interpret the adverb.

(50) The pressure was rapidly low.

Examples like (51), on the other hand, provide a motivation for representing the initial active interval of a transition event (cf. §4.2.3), since the adverb essentially selects for such an interval.

(51) The engine quickly failed.

Durational adverbials like ***for X***, where ***X*** is a temporal measure phrase, modify any interval, but not their endpoints. Finally, relational adverbs, which specify temporal sequence, modify the Reference Time of situations.

Adverbials can combine relational and durational elements. ***In X*** where ***X*** is a temporal measure phrase not only specifies a duration, but also relates the endpoint of this duration to some other time, e.g., the time at which the utterance is produced, as in (52).

(52) The lights will go off in 10 minutes (e.g., ***from now***).

Temporal connectives like ***before*** and ***after*** can combine with temporal measure phrases to yield complex adverbials specifying both a duration and a relation, as in (53).

(53) The engine seized five minutes before the alarm sounded.

In this section, we will look briefly at the two types of durational phrases compared by Vendler [Vendler67] in order to demonstrate the advantages of the representations developed here for interpreting them. Then we will look briefly at the algorithm for interpreting complex sentences with subordinate adverbial clauses.

5.1. Durational Adverbials

Unbounded situations. Predications denoting states and processes have duration, as shown by the interpretation of durational adverbial phrases of the form ***for X***, where ***X*** is a time measure, as in (54) and (55)

(54) The pressure was low for 10 minutes. (state)

(55) The gear was turning for 10 minutes. (unbounded process)

However, as noted in preceding discussions, the past tense in reference to states and unbounded processes does not apply to the whole duration. It applies to

the moment within the interval designated as the situation's Event Time. Since in (54) and (55) the Event Time is past and the Speech Time is present, the two temporal indices create an explicit temporal extent within which to locate the durational phrases. The *for* adverbial phrase also evokes an unbounded duration, meaning that the measure phrase does not necessarily encompass the entire duration, as shown by the lack of contradiction in asserting the continuation of the interval up to the present, as in (56) and (57).

(56) The pressure was low for ten minutes and is still low.

(57) The gear was turning for ten minutes and is still turning.

The present perfect would allow one to assert something semantically very similar to (56) and (57), but more laconically (e.g., ***The pressure has been low for ten minutes***). However, a context in which (56) would be more correct than the corresponding perfect is perfectly possible; it would have to be a context where the pressure is now low, was low over some interval of 10 minutes duration, but where this interval is more than 10 minutes prior to the present, and where the pressure has continued to be low up to the present (e.g., ***A: The alarm should go off if the pressure is low for 10 minutes. B: Well, the pressure was low for 10 minutes and it's still low, but the alarm still hasn't gone off***).

The past tense with an unbounded interval evokes a span of time between the past Event Time and the present Speech Time within which to situate the measure of time given by a *for* adverbial. However, there is no such span of time associated with the present tense of an unbounded interval, hence the impossibility of a *for* measure phrase in examples (58) and (59).[23]

(58) ? The pressure is low for 10 minutes.

(59) ? The gear is turning for 10 minutes.

Note that the present perfect, like the simple past, does provide a temporal point prior to the present, thereby creating a span of time for the durational phrase to apply to, as in (60) and (61).

(60) The pressure has been low for three hours.

(61) The gear has been turning for 5 minutes.

The temporal structures generated for examples like (56)-(59) make it possible to correctly interpret the adverbial phrases they contain. The measure phrases in (56) and (57) can be interpreted not simply because the mentioned situations have duration, but more importantly because of the distinctness of the two temporal indices, Event Time and Speech Time. In (58) and (59), where Event Time and Speech Time coincide, there is no explicit span of time within which to situate the measure phrase. Cases where there is no explicit component of temporal structure in the situation representation to match up with the temporal structure evoked by a temporal adverbial are probably

[23] Since isolated sentences can generally be given a variety of readings, it is often necessary to add qualifications regarding the intended reading of linguistic examples. Sentences like (56) and (57), for example, can be interpreted as ***the pressure is to be low for ten minutes***. Such interpretations are outside the scope of this paper, for they pertain to hypothetical rather than an actual times.

candidates for the kind of coercion discussed in Moens and Steedman (this volume).

The durational adverbial phrases in (62)-(64) not only specify a duration, but also an endpoint [Vendler67]. Since progressive process predications are unbounded, there is no actual endpoint to be mapped to, hence, under one reading, (62) cannot be interpreted as a situation with an actual time; rather, it seems to refer to an activity that was supposed to take place 5 minutes from some time previously specified in the discourse context (e.g., paraphrasable as ***It was to be the case that the gear would turn five minutes from the present***). There is another possible reading, paraphrasable as ***It turned out to be the case that the gear turned five minutes after some previously specified time***, as in the context ***I applied some lubricant to the gear and it was turning in five minutes***, which like (58) and (59) above, may be examples requiring coercion.[24] In contrast, examples (63) and (64) can be interpreted as actual situations whose endpoints coincide with the endpoints of the five minute duration.

(62) The gear was turning in 5 minutes.

(63) The gear turned in 5 minutes.

(64) The engine was repaired in 5 minutes.

The two types of durational adverbials behave differently when modifying the different types of temporal structures in ways which tend to confirm the representations proposed here.

5.2. Complex Sentences

The temporal adverbials encountered in the CASREPs domain consisted predominantly of phrases introduced by temporal connectives, e.g., ***when***, ***before*** and ***after*** [Smith81]. The general problem in analyzing the strictly temporal information associated with such connectives is to associate some time evoked by the matrix clause with some time evoked by the complement phrase. In general, connectives are represented as associating the Reference Time of the matrix clause with the Reference Time of the complement. The procedure involved in analyzing the temporal relations specified by a ***before*** adverb (or other temporal connective) has the six steps illustrated in (65) below.

(65) The compressor failed before the pump seized.

Step 1: Analyze semantics of the main clause	***The compressor failed***
Step 2: Find Reference Time of main clause (RT1)	moment([fail1])
Step 3: Recognize temporal adverb	***before***
Step 4: Analyze semantics of subordinate clause	***the pump seized***
Step 5: Find Reference Time of subord. clause (RT2)	moment([seize1])
Step 6: Look up semantic structure of connective	precede(RT1, RT2)

Result: precedes(moment([fail1]), moment([seize1]))

[24] Thanks to Bonnie Webber and Mark Steedman for pointing out the second reading mentioned here.

First, the temporal semantics of the main clause is analyzed. One of the outputs of this analysis is the Reference Time of the main clause, which in this case would be represented as moment([fail1]). Then the time component finds the adverbial phrase ***before the pump seized*** in the constituent list which it recognizes as consisting of a temporal connective (***before***) and a complement. The complement clause is sent to the semantic interpreter [Palmer85] and is returned to the time component for temporal analysis. The fourth step, the temporal analysis of the subordinate clause, yields the information that the Reference Time of the subordinate clause is moment([seize1]). Finally, the time component looks up the predicate structure representing the semantics of the temporal connective. *Before* is represented as a binary predicate—**precedes**—whose first argument is the Reference Time of the main clause and whose second argument is the Reference Time of the complement clause.

Currently, relational adverbs like ***before***, ***after*** and ***when*** are represented as predicates relating the Reference Times of the modified and modifying situations. The procedure for handling temporal connectives assumes a priori that the Reference Times of the syntactically superordinate and subordinate constituents are the required input. In future work, these and other adverbs will be treated more explicitly as semantic predicates with selectional constraints which guide the search for the appropriate components of temporal structure associated with the referents of the relevant constituents.

6. Conclusion

The situation representations presented here model the temporal meaning of inflected verbs by assigning a semantic value to each of four components, the inherent lexical aspect, the tense, and the presence or absence of the perfect and progressive. Two significant advantages to the overall proposal are the simplicity of the algorithm which computes the representations, and the generality of the building blocks used in constructing them. The algorithm accounts for the context dependencies among the four semantic components through a single mechanism, i.e., an appropriate characterization of the Event Time and its relation to the full temporal structure of a state, process or transition event. These temporal structures are composed of intervals which may be active or stative, and which may be bounded, unbounded, or unspecified for boundedness.

The situation representations have certain advantages in and of themselves. For example, the linkage between the components of temporal structure and Dowty's aspect calculus, and the incorporation of a Reichenbachian treatment of tense, make it possible to represent very precisely what predicates hold when. Further, the dual possibility of associating the ***become*** operator either with an unbounded interval or a transition bound between intervals circumvents the so-called imperfective paradox. An additional advantage is the utility of these representations for further processing. The preceding section

illustrated how the three building blocks of the representations, i.e., the notion of persistence of some situation through an interval, kinesis of the situation, and boundedness of the interval, make it possible to interpret accurately three corresponding kinds of temporal adverbials, and to identify those cases where coercion is required. Finally, explicit representation of the Reference Times and Event Times within distinct types of temporal structures should make it possible to account for the differential contribution of situations to narratives and other types of discourse.

Acknowledgements

I was fortunate in having the opportunity to consider the problems of temporal analysis in the context of a congenial work environment with stimulating colleagues and a large, relatively comprehensive text processing system. The members of my group provided much useful criticism and commentary, especially Lynette Hirschman, Deborah Dahl, Martha Palmer and Carl Weir. Bonnie Webber was extremely generous in her encouragement, and offered invaluable suggestions. I also profited from discussions with Mark Steedman, and his careful reading of earlier verions of this paper. Thank you all.

REFERENCES

[Allen83]
James F. Allen, Maintaining Knowledge about Temporal Intervals. ***Communications of the ACM* 26.11**, 1983, pp. 832-43.

[Allen84]
James F. Allen, Towards a General Theory of Action and Time. ***Artificial Intelligence* 23.2**, 1984, pp. 123-54.

[Austin77]
J. L. Austin, ***How to Do Things with Words.*** Harvard University Press, Cambridge, MA, 1977.

[Comrie76]
Bernard Comrie, ***Aspect: An Introduction to the Study of Verbal Aspect and Related Problems.*** Cambridge University Press, Cambridge, 1976.

[Dahl86]
Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[Dahl87a]
Deborah A. Dahl, John Dowding, Lynette Hirschman, Francois Lang, Marcia Linebarger, Martha Palmer, Rebecca Passonneau, and Leslie Riley, Integrating Syntax, Semantics, and Discourse: DARPA Natural Language Understanding Program, R&D Status Report, Paoli Research Center, Unisys Defense Systems, Paoli, PA, 1987a.

[Dahl87b]
Deborah A. Dahl, Martha S. Palmer, and Rebecca J. Passonneau, Nominalizations in PUNDIT, Proceedings of the 25th Annual Meeting of the ACL, Stanford, CA, 1987b.

[Dowding87]
John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, 2nd International Workshop on Natural Language Understanding and Logic Programming, Vancouver, B.C., Canada, 1987.

[Dowty79]
David Dowty, ***Word Meaning and Montague Grammar.*** D. Reidel, Dordrecht, 1979.

[Dowty82]
David Dowty, Tenses, Time Adverbials and Compositional Semantic Theory. ***Linguistics and Philosophy* 5**, 1982, pp. 23-55.

[Dowty86]
David Dowty, The Effects of Aspectual Class on the Temporal Structure of Discourse: Semantics or Pragmatics. ***Linguistics and Philosophy* 9**, 1986, pp. 37-61.

[Foley84]
William Foley and R. Van Valin, ***Functional Syntax and Universal Grammar***. Cambridge University Press, Cambridge, 1984.

[Freed79]
Alice Freed, ***The Semantics of English Aspectual Complementation***. Reidel, Dordrecht, 1979.

[Jackendoff87]
Ray Jackendoff, The Status of Thematic Relations in Linguistic Theory. ***Linguistic Inquiry* 18.3**, 1987, pp. 369-411.

[Jakobson57]
Roman Jakobson, Shifters, Verbal Categories and the Russian Verb. In ***Selected Writings***, Mouton, The Hague, 1963 (originally 1957).

[Lamiroy87]
Beatrice Lamiroy, The Complement . ion of Aspectual Verbs in French. ***Language* 63.2**, 1987, pp. 278-98.

[Levin86]
Beth Levin and Malka Rappaport, The Formation of Adjectival Passives. ***Linguistic Inquiry* 17**, 1986, pp. 623-62.

[Linebarger88]
Marcia Linebarger, Deborah A. Dahl, Lynette Hirschman, and Rebecca J. Passonneau, Sentence Fragments Regular Structures. ***26th ACL***, 1988.

[McCawley71]
James McCawley, Tense and Time Reference in English. In ***Studies in Linguistic Semantics***, Charles J. Fillmore and D. Terence Langendoen. (ed.), Holt, Rinehart and Winston, New York, 1971, pp. 97-114.

[Moens87]
Mark Moens and Mark J. Steedman, Temporal Ontology in Natural Language. ***Proceedings of the 25th Annual Meeting of the ACL***, 1987.

[Mourelatos81]
Alexander P. D. Mourelatos, Events, Processes and States. In ***Tense and Aspect***, P. J. Tedeschi and A. Zaenen (ed.), Academic Press, New York, 1981, pp. 191-212.

[Mufwene84]
Salikoko S. Mufwene, ***Stativity and the Progressive***. Indiana University Linguistics Club, Bloomington, IN, 1984.

[Newmeyer75]
Frederick Newmeyer, ***English Aspectual Verbs***. Mouton, The Hague, 1975.

[Palmer85]
Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[Palmer86]
Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Proceedings of the 24th ACL, Columbia University, New York, August 1986.

[Passonneau86]
Rebecca J. Passonneau, Designing Lexical Entries for a Limited Domain, Logic-Based Systems Technical Memo No. 42, Paoli Research Center, System Development Corporation, Paoli, PA, April, 1986.

[Reichenbach47]
Hans Reichenbach, ***Elements of Symbolic Logic***. The Free Press, New York, 1947.

[Roberts85]
Craige Roberts, Modal Subordination and Anaphora in Discourse. ***Paper presented at the 60th LSA***, 1985.

[Smith81]
Carlota S. Smith, Semantic and Syntactic Constraints on Temporal Interpretation.. In ***Tense and Aspect***, P. J. Tedeschi and A. Zaenen (ed.), Academic Press, New York, 1981, pp. 213-37.

[Smith86]
Carlota S. Smith, A Speaker-based Approach to Aspect. ***Linguistics and Philosophy*** **9**, 1986, pp. 97-115.

[Talmy85]
Leonard Talmy, Lexicalization Patterns: Semantic Structure in Lexical Forms. In ***Language Typology and Syntactic Description: Grammatical Categories and the Lexicon***, Cambridge University Press, Cambridge, 1985, pp. 63-163.

[Taylor77]
Barry Taylor, Tense and Continuity. ***Linguistics and Philosophy*** **1**, 1977, pp. 199-220.

[Vendler67]
Zeno Vendler, Verbs and Times. In *Linguistics in Philosophy*, Cornell University Press, New York, 1967, pp. 97-121.

[Vlach81]
Frank Vlach, The Semantics of the Progressive. In *Tense and Aspect*, P. J. Tedeschi and A. Zaenen (ed.), Academic Press, New York, 1981, pp. 271-92.

[Webber87]
Bonnie Webber, The Interpretation of Tense in Discourse. *Proceedings of the 25th ACL*, 1987.

Passonneau, Rebecca. "Situations and Intervals". *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, CA., July, 1987.

Situations and Intervals[1]

Rebecca J. Passonneau
March 6, 1987

Knowledge Systems[2]
Defense Systems, UNISYS
P.O. Box 517
Paoli, PA 19301
USA

Summary

The PUNDIT system processes descriptions of situations and the intervals over which they hold using an algorithm that integrates *tense logic* and *aspect*. It analyzes the main verb and its tense, taxis and grammatical aspect to generate representations of three types of situations: states, processes and events. Each type has a distinct temporal structure, represented in terms of intervals having two features: stativity vs. activity, and boundedness.

Topic Area: Temporal Semantics

[1]This work was supported by DARPA under contract N00014-85-C-0012, administered by the Office of Naval Research APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

[2]Formerly Paoli Research Center, SDC—A Burroughs Company

1. Introduction

This paper describes a semantics of situations and the intervals over which they hold that is neither situation semantics (Barwise and Perry, 1983) nor interval semantics (Dowty, 1979, 1982, 1986; Taylor, 1977). It is unfortunately difficult to avoid the overlap in terminology because what will be described here shares with situation semantics the starting point that predications refer to situations. It shares with interval semantics the assumption that the times involved in references to situations are intervals. The concerns addressed here, however, arise from the computational task of processing descriptions of situations in natural language text in order to represent what predicates are asserted to hold over what entities and when.

The PUNDIT text-processing system[3] processes references to situations using an algorithm that integrates *tense logic* (Reichenbach, 1947) with *aspect*, or what Talmy (1985) calls the "pattern of distribution of action through time." The algorithm (Passonneau, 1986) first analyzes the temporal semantics of the main verb of a sentence (i.e., its lexical aspect) and its concommitant categories of tense, taxis[4] and grammatical aspect.[5] This analysis provides a foundation for the subsequent interpretation of temporal adverbials by explicitly representing the components of time which can be modified. This paper describes how PUNDIT represents the temporal structure of three types of situations, namely states, processes and events, how these situations are located in time, and what the computational advantages of these representations are for interpreting temporal adverbials.

Three specific goals for processing references to situations were identified. The first was to distinguish references to actual time, i.e., to specific situations that are said to have

---

[3]PUNDIT is an acronym for *Prolog UNderstanding of Integrated Text*. It is a *modular system*, implemented in Quintus Prolog, with distinct syntactic, semantic and pragmatic components (cf. Dahl, 1986, Palmer et al., 1986).

[4]*Taxis* (Jakobson, 1957) refers to the semantic effect of the presence or absence of the perfect auxiliary.

[5]Aspect is both part of the inherent meaning of a verb (lexical aspect) and also signalled by the presence or absence of the progressive suffix *-ing* (grammatical aspect).

occurred or to be occurring, as in 1),[6]

1) Oil pressure has been slowly decreasing.

from references to types of situations which have not occurred, might occur, or tend to occur, as in 2).

2) The lube oil pump seizes.

This distinction helps determine, among other things, what proposition tense applies to. In 2), the present tense does not pertain to a specific event of the pump seizing, but rather to the tendency for this type of situation to recur. Currently, PUNDIT only processes references to specific situations associated with actual time.

The second processing goal was to closely link the times at which situations hold with the lexical decompositions of the predicates used in referring to situations. This allows PUNDIT to represent precisely what kinds of situations entities participate in and when. The basic components of time are intervals while the two features associated with them are stativity versus activity, and boundedness. The former feature pertains to the internal structure of an interval while boundedness pertains to the way in which the temporal structure associated with a situation is located in time by tense and taxis (cf. § 3.2 below).

The final goal was to represent the times for which situations hold in a sufficiently rich manner to process temporal adverbials modifying different temporal components. For example, both 3) and 4) contain an *at* phrase locating a situation with respect to a clock time.

3) Pressure was low at 08:00.
4) The pump seized at 08:00.

But, the relation of the clock time to the two situations differs. In 3), 08:00 occurs within an interval over which the state of *pressure being low* holds. In 4), it coincides with a transition

[6]PUNDIT's current application is to process short messages texts called CASREPS (CASualty REPorts) which describe Navy equipment failures. As the examples illustrate, the texts often contain sentence fragments.

from a process of *the pump becoming seized* to a state of *the pump being seized.*

2. Problems in Computing Appropriate Representations

The critical problems in the semantic analysis of references to situations and their associated times are: 1) language encodes several different kinds of temporal information, 2) this information is distributed in many distinct linguistic elements, and finally, 3) the semantic contribution of many of these elements is context-dependent and cannot be computed without looking at co-occurring elements.

These problems have been addressed as follows. A decision was made to focus on the kinds of temporal information pertaining to the goals outlined in the previous section and to temporarily ignore other kinds of temporal information.[7] Computation of this information was then divided into relatively independent tasks, with appropriate information passed between tasks to accommodate context-sensitivities. First, the linguistic input which is both syntactically obligatory and semantically critical is computed, i.e., the verb and its categories. This happens in two stages with the aspectual information (lexical and grammatical aspect) computed first and the relational information (tense and taxis) computed second. Since adverbial modification is optional, temporal adverbs are processed not only separately from but also subsequent to the interpretation of the verb and its categories.

3. Solution: Intervals and their Features

3.1. Temporal Structure

The input used to compute the temporal structure of a situation consists of the lexical aspect of the verb and its grammatical aspect. Situations are represented as predicates identifying the type of situation as a state, process or event; they take three arguments: a unique

[7] E.g., rate (given by adverbs like *rapidly*), "patterns of frequency or habituation" (cf. Mourelatos, 1981), and so on

identifier of the situation, the semantic decomposition, and the time argument. Example 6) shows a simple stative sentence, the type of temporal structure it evokes, its semantic decomposition, and a representation of the situation it denotes. The same pointer ([low1]) identifies both the situation and its time argument because it is the actual time for which a situation holds which uniquely identifies it as a specific situation.

6) Sentence: The pressure is low.
Temporal structure: unbounded stative interval
Decomposition: lowP(patient([pressure1]))
Situation: state([low1], lowP(patient([pressure1])), period([low1]))

As shown, stative predications hold over **unbounded stative intervals**. Stative intervals are represented as a **period** time argument to a **state** representation.

Interval semantics captures the distinct temporal properties of situations by specifying a single truth conditional relation between a predication and a unique interval. In contrast, PUNDIT associates two types of features with the intervals over which the predications hold. The feature of **stativity** is defined here just as stative predications are defined in interval semantics: *A sentence ψ is stative iff it follows from the truth of ψ at an interval I that ψ is true at all subintervals of I* (Dowty, 1986, p. 42). But stative predications are also defined in terms of the feature of boundedness, which pertains to how stative intervals participate in **temporal ordering** relations, as will be shown below (§§ 3.2,3.3).

A process is a situation which holds over an active interval of time. Active intervals are represented as a **period** time argument to a **process** representation. Again, the criterion for defining the internal structure of an active interval is borrowed from interval semantics: *A sentence ψ is an activity iff it follows from the truth of ψ at an interval I that ψ is true at all subintervals of I down to a certain limit in size* (Dowty, 1986, p. 42). But in addition, it is proposed that active intervals can be **unbounded** or **unspecified** for boundedness, depending on the grammatical aspect of the predication; both progressive and non-progressive process predications have the

same situational representation (cf. 7a, b).

7)	Sentence:	a) The pump is operating.	b) The pump operated.
	Grammatical Aspect:	**progressive**	**non-progressive**
	Temporal structure:	**unbounded active interval**	**unspecified active interval**
	Decomposition:	operateP(actor([pump1]))	
	Situation:	process([operate1], operateP(actor([pump1])), period([operate1]))	

But the different values of boundedness lead to different relations between **event time** and temporal structure, as will be shown below.

Transition event verbs denote a transition to a new situation. Following Dowty (1979), their decompositions contain the aspectual operator **becomeP**. The aspectual operator's argument is the predicate denoting the situation which results from a transition event. PUNDIT's representations of transition events differ from other proposals[8] by linking distinct time arguments to distinct components of the semantic decomposition. A transition event consists of a process (of **becoming**) leading up to a new state or process. Its temporal structure is thus an active interval followed by--and bounded by--a new active or stative interval (cf. 8).

8)	Sentence:	The engine seized.
	Temporal structure:	**active interval + transition bound + stative interval**
	Decomposition:	becomeP(seizedP(patient([engine1])))
	Situations:	event([seize1], becomeP(seizedP(patient([engine1]))), moment([seize1]))
		state([seize2], seizedP(patient([engine1])), period([seize2]))
	Temporal relation:	starts(moment([seize1]), period([seize2]))

Currently, PUNDIT does not explicitly represent the initial process.[9] The transition is represented as an **event** with a **moment** time argument (e.g., moment([seize1])), and the resulting **state** is represented with a **period** time argument (e.g., period([seize2])) which **starts** at the moment of transition. A transition bound (e.g., moment([seize1]) is an abstract feature rather than a real component of time. It is represented because it participates in temporal ordering relations and can be directly modified by temporal adverbials. It can be thought of as the same kind of

[8] There are some similarities between my *transition bound* and the notion of *nucleus* proposed by Steedman and Moens, Steedman, personal communication.

boundary between intervals implied by Allen's **meets** relation (Allen, 1983; 1984, esp. p. 128).

3.2. Event Time

PUNDIT employs a Reichenbachian analysis of tense which temporally locates situations in terms of three abstract times: the time of the situation (**event time**), the time of speech/text production (**speech time**), and the time with respect to which relational adverbials are interpreted (**reference time**). Reichenbach (1947) did not distinguish between the temporal structure of a situation and its **event time**. For PUNDIT, the **event time** is an abstract component of temporal structure in terms of which ordering relations are specified. It is determined on the basis of boundedness, and is always represented as a dimensionless **moment**.

The three values of boundedness outlined above correspond to three possible relations of **event time** to a time argument. Examples 9) through 11) illustrate these relations. If an interval is unbounded (as in 6 and 7a above), its **event time** is represented as an arbitrary moment included within the interval:

9) The pressure is low.
Event time: M1 *such that* **includes**(period([low1]),moment([M1]))

For an interval unspecified for boundedness (as in 7b above) the event time has a non-committal relation to the interval, i.e., it may be an endpoint of or included within the interval:

10) The pump operated.
Event time: M1 *such that* **has**(period([operate1]),moment([M1]))

The moment time argument of a transition event is its event time:

11) The pump seized.
Event time: moment([seize1])

Defining these three different relations of event time to temporal structure simplifies the computation of the ordering relations given by the perfect and non-perfect tenses.

[6] This is simply a matter of convenience given the needs of the present application domain.

3.3. Temporal Ordering Relations

The **event time** and the verb's tense and taxis comprise the input used in computing temporal ordering relations. The different relations of **event time** to the temporal structures of situations captures several important facts about the interaction of tense and aspect. For example, only unbounded intervals allow present tense, thus present is computed for examples like 6) and 7) above, but not for examples like 2).

Also, a predication denoting a past unbounded situation can be followed by a predication asserting the continuation (or cessation) of the same situation:

12) The pump was operating at 08:00 but is no longer operating.

This is provided for by representing the event time for 12) as a moment included within an interval of indeterminate duration. A similar assertion following a past transition event predication is contradictory:[10]

13) ?The pump sheared the drive shaft and is still shearing it.

The event time for the first conjunct in 13) is a transition bound necessarily culminating in a new situation (i.e., a state of *being sheared*). Since the transition itself is dimensionless, the second conjunct cannot refer to its persistence. A predication evoking an unspecified interval in a similar context can be interpreted analogously to either 12) or 13):

14) The pump operated at 08:00 and is still operating.

The non-committal relation of event time to temporal structure for such cases makes both interpretations possible. Assigning a more specific interpretation is probably pragmatic rather than semantic in nature. As we will see next, the utility of distinguishing between unbounded and unspecified process predications is especially apparent in contexts containing temporal adverbials.

---

[10]The contradiction arises because one naturally interprets *still* as indicating persistence of the same event. One could interpret such a sentence without contradiction as indicating the iteration of the same type of event.

3.4. Conclusion: Adverbial Modification

The representations described above were inspired by remarks found in the literature on tense and aspect (cf. esp. Bull, Dowty, Mourelatos, Vendler) to the effect that "the time schemata" (Vendler, p. 98) associated with different situations are crucial to the way we perceive and talk about them. One of the crucial types of evidence used in deriving the representations was the interpretation of temporal adverbials in different contexts. Consequently, one of the advantages to the representations is that they make it possible to tailor the interpretation of a temporal adverb to the temporal structure of the modified situation.

For example, specifying a different relation for the **event time** of an active interval, depending on grammatical aspect, yields different temporal relations between the situations described in sentences like 14)-16).[11]

14) The pump failed when the engine was rotating.
transition of failure during period of rotation

15) The pump failed when the engine rotated.
transition of failure during OR at one endpoint of period of rotation

16) The engine rotated when the pump failed.
Same as 15)

Sentences like 16) are often, but not always, interpreted with the process (e.g., *rotation*) beginning at or after the transition event moment (e.g., *failure*). PUNDIT's representations of the temporal semantics of predications are explicit enough yet sufficiently non-committal to provide suitable input to a pragmatic reasoner that could decide these cases.

Acknowledgements

I would like to thank several people for their comments, encouragement and patience: Martha Palmer, Lynette Hirschman, Bonnie Webber and Debbie Dahl.

[11] The different relations are shown informally in the examples; the formal representations PUNDIT generates will be given in the full paper.

References Cited

Allen, James F. 1984. Towards a general theory of action and time. Artificial Intelligence 23.2: 123-154.

Allen, James. F. 1983. Maintaining knowledge about temporal intervals. Communications of the ACM 26.11:832-843.

Barwise, Jon and John Perry. 1983. Situations and Attitudes. Cambridge, Massachusetts: The MIT Press.

Bull, William E. 1971. [1963]. Time, Tense and the Verb. University of California Publications in Linguistics 19. Berkeley: University of California Press.

Dahl, Deborah A. 1986. Focusing and Reference Resolution in PUNDIT. Presented at AAAI, Philadelphia, PA.

Dowty, David R. 1986. The effects of aspectual class on the temporal structure of discourse: semantics or pragmatics? Linguistics and Philosophy 9: 37-61.

Dowty, David R. 1982. Tense, time adverbials and compositional semantic theory. Linguistics and Philosophy 5: 23-55.

Dowty, David R. 1979. Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's PTQ. Dordrecht: D. Reidel.

Jakobson, Roman. 1971 [1957]. Shifters, verbal categories and the Russian verb. In his Selected Writings, Vol. 2, pp. 130-147. The Hague: Mouton.

Mourelatos, Alexander P. D. 1981. Events, processes, and states. In Syntax and Semantics, vol 14: *Tense and Aspect*, pp. 191-212. *Edited* by P. J. Tedeschi and A. Zaenen. New York: Academic Press.

Palmer, Martha; Dahl, Deborah A.; Schiffman, Rebecca J. [Passonneau]; Hirschman, Lynette; Linebarger, Marcia; Dowding, John. 1986. Recovering Implicit Information. 24th Annual Meeting of the Association for Computational Linguistics. Columbia University, New York.

Reichenbach, Hans. 1947. Elements of Symbolic Logic. New York: The Free Press.

Passonneau, Rebecca. 1986. A Computational Model of the Semantics of Tense and Aspect. Logic-Based Systems Technical Memo No. 43. Paoli Research Center. System Development Corporation. December, 1986.

Talmy, Leonard. 1985. Lexicalization patterns: semantic structure in lexical forms. In Language Typology and Syntactic Description, vol. 3: Grammatical Categories and the Lexicon, pp. 57-151. Edited by Timothy Shopen. Cambridge: Cambridge University Press.

Taylor, Barry. 1977. Tense and continuity. Linguistics and Philosophy 1.

Vendler, Zeno. 1967. Verbs and times. Linguistics in Philosophy. New York: Cornell University Press.

Vlach, Frank. 1981. The Semantics of the progressive. In Syntax and Semantics, vol 14: Tense and Aspect, pp. 271-292. Edited by P. J. Tedeschi and A. Zaenen. New York: Academic Press.